UNIVERSIDAD POLITÉCNICA DE MADRID

Diseño de nuevas técnicas evolutivas para videojuegos de lucha

Tesis fin de Máster

Departamento de Inteligencia Artificial
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Autor: Carlos Javier López Turiégano
Tutores: Daniel Manrique Gamo, José María Font Fernández

23/07/2015

Resumen

El objetivo de esta tesis fin de máster es la construcción mediante técnicas evolutivas de bases de conocimiento con reglas difusas para desarrollar un sistema autónomo que sea capaz de jugar con destreza a un videojuego de lucha en 2D.

El uso de la lógica difusa permite manejar imprecisión, la cual está implícita en las variables de entrada al sistema y favorece la comprensión a nivel humano del comportamiento general del controlador.

Se ha diseñado, para obtener la base de conocimiento que permita al sistema tomar las decisiones adecuadas durante el combate, un nuevo operador para algoritmos evolutivos. Se ha observado que la programación genética guiada por gramáticas (PGGG) muestra un sesgo debido al cruce que se suele emplear para obtener nuevos individuos en el proceso evolutivo. Para solventar este problema, se propone el método de sedimentación, capaz de evitar la tendencia que tiene la PGGG a generar bases de conocimiento con pocas reglas, de forma independiente a la gramática. Este método se inspira en la sedimentación que se produce en el fondo de los lechos marinos y permite obtener un sustrato de reglas óptimas que forman la solución final una vez que converge el algoritmo.

Abstract

The objective of this thesis is the construction by evolutionary techniques of fuzzy rule-base system to develop an autonomous controller capable of playing a 2D fighting game.

The use of fuzzy logic handles imprecision, which is implicit in the input variables of the system and makes the behavior of the controller easier to understand by humans.

A new operator for evolutionary algorithms is designed to obtain the knowledge base that allows the system to take appropriate decision during combat. It has been observed that the grammar guided genetic programming (GGGP) shows a bias due to the crossing that is often used for obtaining new individuals in the evolutionary process. To solve this problem, the sedimentation method, able to avoid the tendency of the PGGG to generate knowledge bases with few rules, independently of the grammar is proposed. This method is inspired by the sedimentation that occurs on the bottom of the seabed and creates an optimal rules substrate that ends on the final solution once the algorithm converges.

Tabla de contenidos

Lista de figuras

Lista de tablas

1 Introducción

La Computación Natural (CN) es un área de investigación multidisciplinar que incluye aspectos relacionados con la biología, la física, la química y el desarrollo de algoritmos matemáticos y programas informáticos [KARI08]. Sus principales metas son el desarrollo de modelos de computación inspirados en la naturaleza y la comprensión del funcionamiento del mundo natural en términos de procesamiento de información. Dentro de la CN se pueden distinguir dos tendencias de investigación que guardan relación con esta tesis: la primera de ellas se caracteriza por el desarrollo sistemas biológicos sintéticos cuyo fin es la comprensión de diversos fenómenos naturales partiendo del entendimiento de sus partes constituyentes; mientras que la segunda engloba a las técnicas que toman la naturaleza como base para desarrollar modelos de computación bio-inspirados, donde se encuadra la computación evolutiva.

La Inteligencia Artificial (IA) ha estudiado ampliamente la representación del conocimiento. Su principal objetivo es la creación de formalizaciones del conocimiento que sean procesables por programas informáticos. El modelo de razonamiento humano sirve de inspiración para crear sistemas de representación del conocimiento que aborden problemas de la misma forma que lo haría la mente humana. Los sistemas basados en reglas lingüísticas constituyen una técnica capaz de servir al propósito de imitar el modo de razonamiento humano en un ordenador [PAJA05].

La Computación Evolutiva (CE) es una técnica representativa de los sistemas bio-inspirados. Un sistema de CE parte de una población de individuos que codifican soluciones a un problema determinado. Esta población es evolucionada mediante operadores de selección, cruce, mutación y reemplazo hasta obtener el individuo que codifique la mejor solución para el problema [MORA08]. La aplicación de técnicas de CE muestra resultados satisfactorios resolviendo problemas de optimización y búsqueda de soluciones, como la generación automática de sistemas inteligentes de propósito específico de diversa índole [FONT12].

Este trabajo muestra aúna un sistema de CE y los sistemas basados en reglas para solucionar un problema donde interviene la estrategia y la habilidad. Asimismo, presenta un método evolutivo que, inspirado en la sedimentación en los lechos marinos permite obtener bases de conocimiento óptimas para jugar con destreza a un videojuego de lucha 2D.

Se organiza el presente trabajo en capítulos cuyo orden y contenido son expuestos a continuación:

- **Capítulo 1:** Introducción.

- **Capítulo 2:** Descripción de los lenguajes y las gramáticas formales, los sistemas basados en reglas y la lógica difusa.

- **Capítulo 3:** Exposición de la computación evolutiva, detallando sus características y las vertientes referidas a los algoritmos genéticos y la programación genética.

- **Capítulo 4:** Análisis de las técnicas de inteligencia artificial aplicadas a los videojuegos de lucha y la lógica difusa aplicada en diferentes géneros de videojuegos.

- **Capítulo 5:** Planteamiento del problema de la generación de una base de conocimiento con reglas difusas para jugar un videojuego de lucha 2D aplicando técnicas de computación evolutiva.

- **Capítulo 6:** Exposición del sesgo observado en la PGGG con cruce Whigham en función de la gramática empleada. Presentación y descripción del método evolutivo basado en la sedimentación que carece de sesgo dependiente de la gramática.

- **Capítulo 7:** Resultados obtenidos de la aplicación del método de la sedimentación en comparación con la PGGG empleada con dos gramáticas con diferentes diseños que muestran la influencia de las características de la gramática utilizada en los resultados obtenidos.

- **Capítulo 8:** Conclusiones y líneas futuras de investigación obtenidas a partir de los resultados del presente trabajo.

2 Sistemas basados en reglas difusas

La representación del conocimiento ha sido ampliamente estudiada por la Inteligencia Artificial (IA). Su principal objetivo es la creación de formalizaciones del conocimiento que sean procesables por programas informáticos y que se acerquen al modelo de razonamiento humano. Las reglas lingüísticas constituyen una técnica capaz de ejercer de interfaz entre el modo de razonamiento humano y el de procesamiento de un ordenador [PAJA05]. Estas reglas hacen uso de términos lingüísticos y siguen una estructura condicional compuesta por antecedentes y consecuentes, conformando sentencias condicionales del tipo "si X entonces Y", fácilmente comprensibles por un ser humano a la par que procesables por un algoritmo. Estas reglas también se conocen como reglas de inferencia debido a su capacidad de inferir conclusiones a partir del conocimiento actual del entorno. La mayoría de los problemas del mundo real tienen imprecisión de forma implícita, tanto en los datos de entrada como en el resultado inferido. La lógica difusa permite representar esta imprecisión propia de los problemas a tratar sin perder la capacidad de ser procesables de forma algorítmica.

Las aplicaciones de los sistemas de representación del conocimiento basados en reglas difusas son variadas y usualmente se ubican dentro de los problemas del mundo real: medicina [GOET95], agricultura [MAHA03] o control de maquinaria [ANIB04].

2.1 Lenguajes y gramáticas

Se define lenguaje formal como un conjunto de palabras, también llamadas sentencias o cadenas, formadas por símbolos de un alfabeto. Una gramática define la estructura de un lenguaje, es decir, las sentencias que lo forman, proporcionando las formas válidas en que se pueden combinar los símbolos del alfabeto. Una consideración importante es la distinción entre lenguajes formales, que son los que se tratan en este capítulo, y lenguajes naturales (inglés, español, etc.). La diferencia estriba en que los lenguajes formales (como los lenguajes de programación) obedecen a reglas preestablecidas y, por tanto, se ajustan a ellas, no evolucionan y han sido creados para un fin específico. Sin embargo, en los lenguajes naturales (utilizados por el hombre), las reglas gramaticales que rigen su estructura han sido desarrolladas con posterioridad para explicar el entorno cambiante.

El concepto de gramática procede de los estudios de Chomsky [CHOM57] en su búsqueda de una descripción formalizada de las oraciones de un lenguaje natural. Chomsky clasificó las gramáticas en cuatro grandes grupos (G0, G1, G2, G3) cada uno de los cuales es un súper conjunto del anterior ($G3 \subset G2 \subset G1 \subset G0$). Las gramáticas tipo 0 se denominan gramáticas sin restricciones o gramáticas de estructura de frases; las gramáticas de Tipo 1 se denominan sensibles al contexto; las gramáticas de Tipo 2 se conocen como gramáticas independientes del contexto o libres de contexto y, por último, las gramáticas de Tipo 3 se denominan gramáticas regulares. Cada tipo añade restricciones al tipo inmediatamente superior y la jerarquía va desde la más general a la más restrictiva. Cada una de estas gramáticas es capaz de generar un tipo de lenguaje. Un lenguaje L se llama del tipo i (i = 0, 1, 2, 3) si existe una gramática G del tipo i capaz de generar o describir ese lenguaje.

Se define un alfabeto Σ como un conjunto no vacío finito de elementos indivisibles denominados símbolos (letras, números o combinaciones de letras y números)**.** Una palabra es una secuencia finita de símbolos de un alfabeto. Se llama longitud de una palabra x (|x|) al número de símbolos que la componen. La palabra cuya longitud es 0, es decir, no contiene ningún símbolo, se conoce como palabra vacía (λ), $| \lambda | = 0$. Se denomina universo de un alfabeto Σ al conjunto de todas las palabras que se pueden formar con los símbolos de dicho alfabeto. Se representa como $W(\Sigma)$. Un lenguaje sobre un alfabeto ($L(\Sigma)$) es un subconjunto de $W(\Sigma)$.

Sea Σ un alfabeto. Se denomina Σ^+ al conjunto de todas las posibles concatenaciones de los símbolos del alfabeto, excluyendo la palabra vacía. Σ^* es la unión de Σ^+ y la palabra vacía λ. Se llama producción o regla a un par ordenado *(x,y)*, donde $x \in \Sigma^+$, $y \in \Sigma^*$. Se dice que *x* es la parte izquierda de la producción e *y* la parte derecha. Es frecuente representar las reglas con la siguiente notación (se lee como x deriva en y, o x produce y):

$$x ::= y$$

Una derivación directa, $v \rightarrow w$, es la aplicación de una producción *(x::=y)* a una palabra *v* para convertirla en otra palabra *w* donde se puede definir $v = z \cdot x \cdot u$, $w = z \cdot y \cdot u$ $(v,w,z,u \in \Sigma^*)$ [ISAS97].

Las gramáticas formales son descripciones estructurales de las sentencias de los lenguajes formales. Las definiciones de los lenguajes formales pueden ser extensionales; es decir, para describir el lenguaje se enumeran todas las palabras que lo forman. Los lenguajes se describen mediante el alfabeto sobre el que se construyen sus palabras; un símbolo inicial del que parte la obtención de cualquiera de las palabras

del lenguaje, denominado axioma; un conjunto de símbolos especiales denominados no terminales, que permiten representar subconjuntos del lenguaje o estados intermedios de la generación de las palabras del lenguaje; y un conjunto de producciones, que permiten realizar las transformaciones desde los símbolos no terminales a las palabras del lenguaje.

Por tanto, una gramática formal *G* se puede definir como una cuádrupla $G = (\Sigma_T, \Sigma_N, S, P)$, donde:

- Σ_T es el alfabeto de los símbolos terminales.
- Σ_N es el alfabeto de los símbolos no terminales, cumpliéndose:

$$\Sigma = \Sigma_T \cup \Sigma_N \text{ y } \Sigma_T \cap \Sigma_N = \emptyset$$

- S es un símbolo no terminal especial ($S \in \Sigma_N$), denominado el axioma de la gramática.
- P es un conjunto finito de reglas (o producciones) que tienen como única restricción que en la parte izquierda de las producciones debe haber al menos un símbolo no terminal, es decir, $P = \{(u::=v) | u = xAy, u \in \Sigma^+, v,x,y \in \Sigma^*, A \in \Sigma_N\}$

Los árboles de derivación representan gráficamente el conjunto de producciones empleadas para generar una palabra desde el axioma. Se pueden utilizar en la construcción de compiladores para representar el análisis sintáctico de los programas fuente, lo que sirve de base para la compilación de código. Sólo se pueden definir árboles de derivación para gramáticas de tipo 1 o más restrictivas (tipos 2 y 3).

Un árbol de derivación se construye de la siguiente manera:

1. La raíz del árbol se denota por el axioma de la gramática.

2. Una derivación directa se representa por un conjunto de ramas que salen de un nodo dado. Al aplicar una regla, un símbolo de la parte izquierda queda sustituido por una palabra (x) de la parte derecha. Por cada uno de los símbolos de x se dibuja una rama que parte del nodo dado y termina en otro, denotado por dicho símbolo.

Sean dos símbolos A y B en la palabra x. Si A está a la izquierda de B en x, entonces la rama que termina en A se dibuja a la izquierda de la rama que termina en B. A lo largo del proceso de construcción del árbol, sus nodos hoja, leídos de izquierda a derecha, forman la palabra obtenida por la derivación representada por el árbol. Se

llama rama terminal a aquella que se dirige hacia un nodo que contiene un símbolo terminal de la gramática. Este nodo se denomina hoja o nodo terminal del árbol.

Por ejemplo, en la gramática definida como:

$$G = (\Sigma_T, \Sigma_N, E, P)$$

$$\Sigma_T = \{0,1,+\}$$

$$\Sigma_N = \{E\}$$

$$P = \{ E ::= E + E, \quad E::= 1, \quad E ::= 0\}$$

La Figura 1 muestra un posible árbol de derivación para obtener la palabra *1+0+1*.

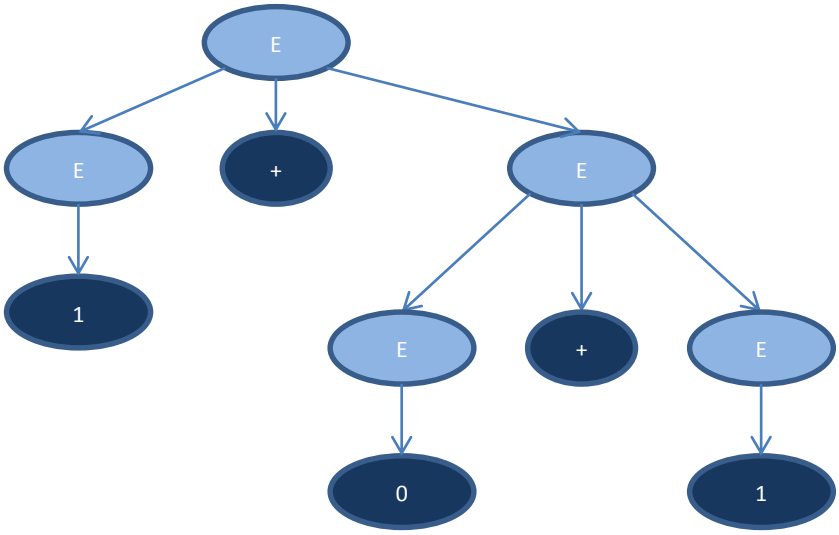


Figura 1: Árbol de derivación de la palabra 1+0+1 según la gramática G

2.2 Sistemas de representación de conocimiento basados en reglas.

Un Sistema Basado en el Conocimiento (SBC) es aquel en el que la representación del conocimiento de un dominio determinado es tal que resulta procesable por un programa informático [PAJA05]. La finalidad de un SBC es la resolución de problemas del dominio para el que ha sido creado, aplicando técnicas de razonamiento sobre el conocimiento que alberga su base de conocimiento (BC). La base de

conocimiento es el pilar central de un SBC, ya que almacena el conocimiento disponible sobre un dominio específico.

En función de cómo se representa el conocimiento acerca de un determinado dominio, la IA se puede dividir en dos áreas principales: la IA simbólica y la IA subsimbólica. La IA simbólica representa el conocimiento empleando lenguajes formales de representación de conocimiento comprensibles por el ser humano, aunque más alejado de la representación de los datos por parte de una máquina. La principal línea de estudio e investigación de la IA simbólica son los SBC, sistemas inteligentes de propósito específico donde lo más importante es obtener soluciones satisfacientes en un determinado dominio. Por el contrario, o de forma complementaria, se encuentra la IA subsimbólica, en donde la forma de representar el conocimiento se aleja más del entendimiento humano. Dentro de esta vertiente se encuentran, entre otras, las redes de neuronas artificiales o la computación evolutiva. En estos casos, además, existe una componente de inspiración en la naturaleza a la hora de construir el sistema inteligente: se simulan los procesos biológicos que ocurren en la naturaleza y que dan como resultado comportamientos que podemos considerar "inteligentes". Las redes de neuronas artificiales tratan de simular el sistema nervioso (o parte de él) animal y la computación evolutiva simula el proceso de evolución de las especies de da como resultado individuos más adaptados al medio de vida. En esta TFM se proporciona una solución simbólica al problema que se plantea, aunque para su búsqueda se emplean técnicas basadas en computación evolutiva.

Una representación simbólica del conocimiento acertada debe ser fácilmente modificable por un programa automático. Esta cualidad permite la depuración del conocimiento almacenado en una base de conocimiento simbólica mediante la incorporación de nuevo conocimiento o la depuración del ya existente. Se dice que una representación simbólica se hace más abstracta a medida que se aleja del trasfondo matemático o conceptual para acercarse al lenguaje natural. Dependiendo del grado de abstracción, se distinguen tres tipos de representaciones simbólicas; procedural y declarativa, relacional y jerárquica [PAJA05]:

La representación procedural es la forma menos abstracta. El conocimiento se almacena en procedimientos enunciados mediante sentencias entendibles por una máquina, de tal forma que la representación del conocimiento y de las instrucciones necesarias para su manipulación se realiza de forma indivisible. La representación declarativa es una extensión de la representación procedural que permite la representación, por separado, del conocimiento, y de las técnicas para su procesado. A la representación simbólica declarativa pertenecen los sistemas basados en reglas o RBSs (del inglés Rule-Based Systems).

La representación relacional se encuentra un escalón por encima de la representación declarativa en términos de abstracción y hace uso de conceptos y relaciones entre conceptos para representar el conocimiento. Este tipo de representación es fuertemente dependiente del tipo de conocimiento que se desea representar. Los conceptos se representan mediante tuplas de información que contienen atributos específicos para almacenar información. Dependiendo del tipo de información con el que se esté tratando, los atributos y conceptos de la representación relacional varían. Las relaciones entre conceptos almacenan información sobre la interdependencia de conceptos. Este modelo de representación es típico de las bases de datos relacionales.

La representación jerárquica surge como una evolución de la representación relacional, aumentando su grado de abstracción y añadiendo un tipo específico de relación entre conceptos: la herencia. Esta relación permite agrupar conceptos con similitud de atributos, los cuales se heredan de unos a otros creando una estructura jerárquica. Los conceptos más elevados dentro de la jerarquía son los más abstractos, siendo los conceptos inferiores de la jerarquía especificaciones de aquellos. Esta amplitud del espectro de abstracción permite a la representación jerárquica procesar información a distintos niveles de profundidad o granularidad. La representación jerárquica es usada por el modelo de representación basada en marcos, así como por los lenguajes de programación orientada a objetos.

Un sistema basado en reglas tiene las siguientes características (Figura 2):

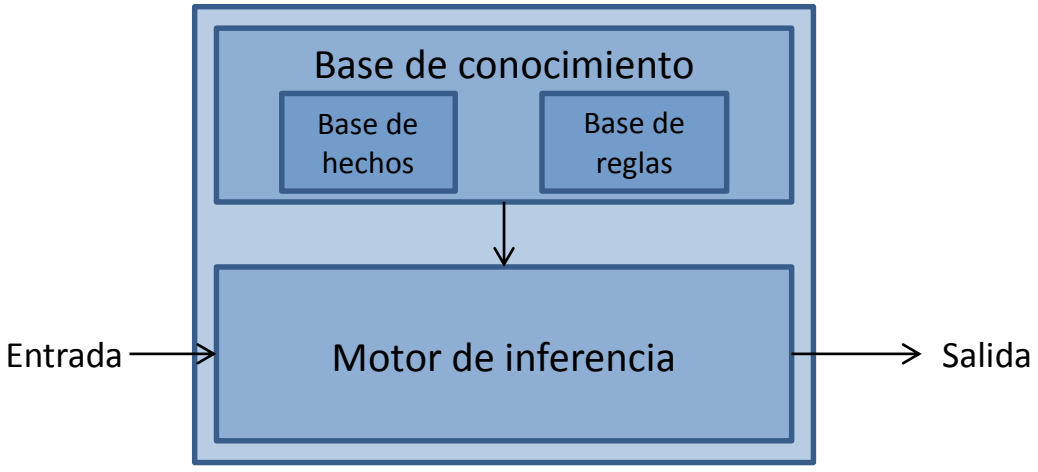


Figura 2 Estructura de un RBS

La base de conocimiento almacena el conocimiento que tiene acerca de un determinado dominio. Se divide en la base de hechos y en la base de reglas. La base

de hechos contiene el estado actual del sistema. La base de reglas almacena las sentencias condicionales donde las variables de entrada se equiparan en el antecedente y las variables de salida son modificadas en el consecuente en el caso de que la regla llegue a aplicarse.

El motor de inferencia utiliza el conocimiento almacenado en forma de hechos y reglas para generar una salida ante una pregunta que se le formula al sistema. Para ello se emplean métodos matemáticos capaces de emplear el formalismo elegido de representación del conocimiento.

2.3 Sistemas de representación de conocimiento basados en reglas difusas.

Los sistemas basados en reglas difusas (FRBSs, de su nombre en inglés Fuzzy Rule-Based Systems) son una extensión de los sistemas clásicos de representación del conocimiento basados en reglas [CORD01]. La diferencia entre ambas reside en que la primera representa el conocimiento impreciso, más parecido a la forma humana de representar su conocimiento [TRIL92].

La lógica difusa empleada para la representación del conocimiento fue descrita por Lofti A. Zadeh [ZADE65]. La representación se realiza mediante variables que toman valores lingüísticos representados mediante conjuntos difusos, las cuales se definen mediante funciones de pertenencia. Estas funciones representan la posibilidad de pertenencia de un valor concreto al conjunto difuso que describe. Si se toma como ejemplo las variables lingüísticas relativas a la temperatura, se pueden definir como tales “Frío”, “Templado” y “Cálido”. Las temperaturas por debajo de 10 grados se consideran frías, las próximas a 20 se consideran templadas y las superiores a 30, cálidas. Según cómo se definan las funciones de pertenencia, la temperatura “15 grados” puede pertenecer a los conjuntos “Frío” y “Templado” al mismo tiempo con mayor o menor posibilidad. La posibilidad de pertenencia parcial a un conjunto es lo que caracteriza a la lógica difusa.

La aplicación de la lógica difusa a los sistemas basados en reglas permite resolver problemas en dominios donde existe imprecisión, que son los más habituales en el mundo real. Se puede representar el conocimiento de forma más cercana a la realidad al poder utilizar las variables lingüísticas propias de los humanos en el sistema en cuestión. La lógica difusa también favorece la puesta en común de conocimiento de

varios expertos en una materia al permitir imprecisión en los umbrales que deben definir estos mismos expertos.

E.H. Mamdani construyó el primer FRBS que, utilizando la formulación proporcionada por Zadeh, aplicó un sistema de lógica difusa a un problema de control [MAMD75]. Esta aplicación de la lógica difusa se conoce como FLC, del inglés Fuzzy Logic Controller o controlador mediante lógica difusa. Su estructura básica, que es muy similar a la de un RBS, se muestra en la Figura 3:

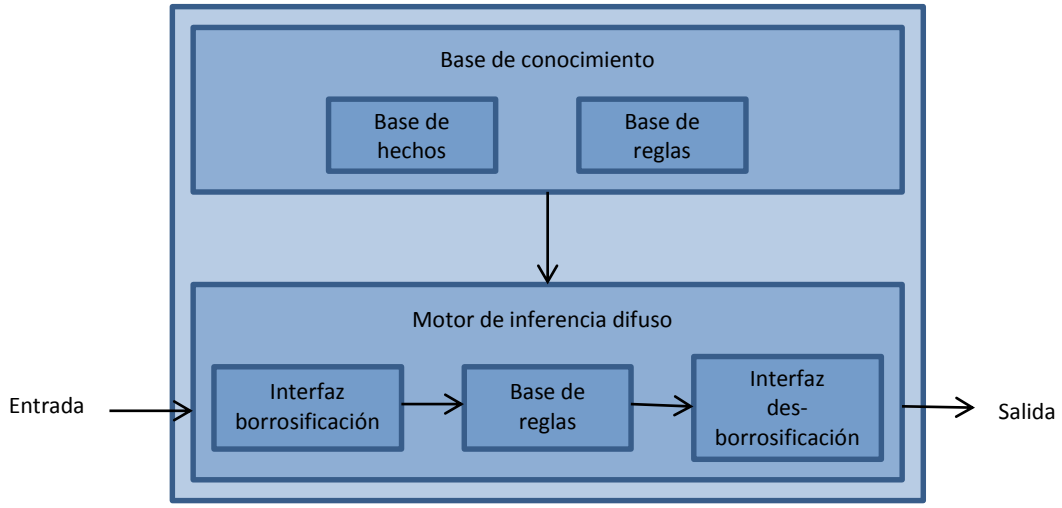


Figura 3 Esquema de un FRBS

La base de conocimiento almacena todo el conocimiento disponible del problema a tratar en forma de hechos y reglas difusas, a través de los cuales opera el sistema de inferencia para generar una salida. Se divide en una base de hechos y en una base de reglas. La base de hechos contiene un conjunto de variables y etiquetas lingüísticas, cada una de las cuales se define mediante un conjunto difuso. Cada conjunto difuso se define mediante una función de pertenencia que indica el grado de pertenencia de un valor a dicho conjunto. La base de reglas es un conjunto de reglas construidas con las variables y etiquetas lingüísticas de la base de hechos, los operadores lógicos AND, OR y NOT, y las partículas condicionales IF, THEN. La parte de una regla situada entre IF y THEN es el antecedente de la regla, siendo el resto el consecuente.

El motor de inferencia consta de una interfaz de *borrosificación*, un sistema de inferencia y una interfaz de *desborrosificación*. La interfaz de *borrosificación* permite al FRBS traducir entradas numéricas a sus correspondientes valores en el universo de los conjuntos difusos, con los que opera el sistema de inferencia. El sistema de

inferencia genera un resultado de acuerdo a las entradas al sistema, a los hechos y reglas almacenados en la base de conocimiento. Este procedimiento se lleva a cabo a través del modus ponens generalizado, que es una extensión del modus ponens de la lógica clásica:

IF X is A THEN Y is B

X is A'

Y is B'

En la lógica borrosa se puede generalizar el modus ponens al no requerir X es A, sino X es A'. De esta forma, la función de pertenencia de $\mu_{B'}(y)$ se calcula con la regla composicional de inferencia (RCI):

$$\mu_{B'}(y) = \mathrm{Sup}_{x \in X} \{T(\mu_{A'}(x), \mu_{A \to B}(x, y))\}$$

donde $\mu_{A \to B}(x, y)$ es una implicación donde previamente se han realizado las extensiones cilíndricas de $\mu_B(y)$ con x y $\mu_A(x)$ con y. El operador T es una T-norma como T(x,y) = min(x,y). *Sup* es el supremo en x de $T(\mu_{A'}(x), \mu_{A \to B}(x, y))$ que está en función de x e y.

Aplicando la RCI a cada regla se obtiene $\mu_{B1'}(y)$, $\mu_{B2'}(y)$, ..., $\mu_{Bn'}(y)$, siendo n el número de reglas de la base de conocimiento. Se obtiene la distribución final $\mu_B(y)$ mediante la t-conorma S:

$$\mu_{B'}(y) = S(\mu_{B1'}(y), \mu_{B2'}(y), \ldots, \mu_{Bn'}(y))$$

Para la *desborrosificación* se emplea el método del centro de gravedad. Este método proyecta en el eje X el centro de gravedad de la distribución como se muestra en la Figura 4.

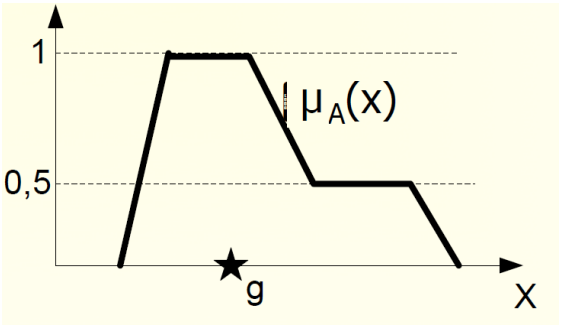


Figura 4 Desborrosificación según el método del centro de gravedad

Para poder realizar este cálculo es necesario muestrear la distribución. Una vez obtenidas las N muestras, se aplica la siguiente fórmula para obtener g, donde x_i son los puntos de muestreo en el eje x:

$$g = \frac{\sum_{i=1}^{N} x_i \cdot \mu_A(x_i)}{\sum_{i=1}^{N} \mu_A(x_i)}$$

3 Computación evolutiva

La imitación del comportamiento de la naturaleza para solucionar diferentes problemas empleando ordenadores se ha ido perfeccionando a medida que la tecnología computacional ha ido madurando. La simulación de la evolución natural [JANG12], el comportamiento de las colonias de hormigas [DORI10] o el movimiento de las bandadas de pájaros [KENN10] son ejemplos de situaciones de la naturaleza llevadas a la computación para la resolución de diferentes problemas.

La Computación Evolutiva (CE) es un paradigma de la Computación Natural que, en lugar de tratar de emular las características de un único organismo biológico, toma como inspiración el funcionamiento de especies enteras de organismos vivos, basándose en la teoría darwiniana de la evolución natural de las especies (Kari & Rozenberg, 2008). Un sistema de CE simula la evolución de una población de individuos, dirigiéndola hacia la consecución de una serie de objetivos previamente designados. Esta evolución simulada se consigue mediante la aplicación de un conjunto de operadores genéticos que posibilitan la creación de nuevas generaciones de individuos, la evaluación de su capacidad para alcanzar los objetivos deseados y la supremacía de aquellos individuos que obtengan una mejor evaluación frente al resto. La computación evolutiva comienza en la década de los años 60. Durante este periodo se acuñan los términos de programación genética (PG) [FOGE66], algoritmos genéticos (AG) [HOLL69] y estrategias evolutivas (EE) [RECH65]. La computación evolutiva permite el desarrollo de diferentes campos como la optimización, la robustez y la inteligencia artificial.

En el libro *El origen de las especies*, Darwin expuso la evolución de las especies como una optimización de los organismos para su supervivencia en medios hostiles [CHAR29]. Es importante destacar que "óptimo" no es sinónimo de "perfecto". La evolución permite obtener una solución altamente satisfactoria para el problema en cuestión pero no garantiza el resultado perfecto. Aún así, este método es mejor que otras técnicas clásicas utilizadas de optimización, como el descenso de gradiente, en problemas con componentes estocásticas o con cierto caos donde no se conoce o no se sabe si hay solución.

En computación, la robustez indica la capacidad de un sistema de adaptarse a un cambio en el propio sistema o en su objetivo. La constante comparativa de resultados de los sistemas evolutivos otorga la capacidad a estos algoritmos de seleccionar las nuevas soluciones en base a los cambios que puedan producirse en cualquier momento.

Las teorías neo-darwinistas proponen como factor determinante de supervivencia la capacidad de aprendizaje de los individuos durante su vida. De esta forma, una especie

cuyos individuos sean más inteligentes tendrán más probabilidades de sobrevivir. Una forma de conseguir inteligencia artificial es mediante evolución para el desarrollo de algoritmos predictivos.

3.1 Características de la computación evolutiva

La computación evolutiva se basa en las teorías neodarwinistas sobre la evolución de las especies a lo largo de la historia. Esta teoría establece como procesos fundamentales la reproducción, la mutación, la competición y la selección. Estos procesos pueden implicar a uno o varios individuos de una población al mismo tiempo [MARC92].
La reproducción es la capacidad de las especies de transmitir su código genético a las siguientes generaciones. Existen dos tipos de reproducción: sexual y asexual. En la primera intervienen dos individuos cuyo resultado es uno o varios individuos cuyo código genético es una mezcla del de los padres. En el caso de la reproducción asexual, sólo es necesario un individuo para llevarla a cabo y el resultado es uno o varios individuos iguales a su progenitor. La reproducción es un proceso que aumenta el número de individuos de una población.
La mutación es el resultado de la aparición de errores en la transcripción del código genético de un progenitor a un descendiente. Este proceso es vital para la adaptabilidad de la especie, especialmente en aquellas cuya reproducción es asexual, pues es la única forma en la que puede cambiar el código genético de la especie. Este proceso no modifica el número de individuos de la población, sino que modifica un único individuo.
La competición se produce debido a las limitaciones del medio. El aumento del número de individuos reduce los recursos disponibles para cada uno de ellos.
La selección es la consecuencia de la combinación de la reproducción y la competición. Los individuos que mejor se adapten al medio y sean capaces de obtener sus recursos necesarios de una manera más eficiente serán los que sobrevivan. Este proceso reduce el número de individuos de una población, equilibrando el sistema.

Las características de los individuos de una especie vienen determinados por su código genético. Este código tiene su impacto en el entorno del individuo. Se pueden diferenciar los términos genotipo para el código genético en sí y el fenotipo para su representación funcional en conjunto en el mundo. Sin embargo, los efectos de la pleitropía y la poligénesis hacen que no exista una relación directa inequívoca entre genotipo y fenotipo. La pleitropía provoca que un único gen pueda afectar a varias características del fenotipo. Por el contrario, la poligénesis establece que una característica del fenotipo sea el resultado de la interacción de varios genes al mismo tiempo. Esto establece que no existe una relación directa uno a uno entre genotipo y fenotipo, sino que la relación es una función no lineal en muchos casos.

Los procesos de reproducción y mutación actúan sobre el genotipo, mientras que los de competición y selección lo hacen según el fenotipo. De esta forma, la evolución es un proceso de optimización de fenotipos donde las variables son genotipos y cuya relación entre ambos no es lineal. Esto favorece la diversidad de genotipos con fenotipos igualmente bien adaptados.

La computación evolutiva plasma lo anteriormente expuesto con diversos ajustes para hacerlo más útil desde el punto de vista conceptual eliminando las restricciones biológicas. Un algoritmo de Computación Evolutiva o Algoritmo Evolutivo (AE), es un proceso iterativo que realiza un conjunto de operaciones de manera reiterada sobre una población de individuos hasta alcanzar ciertas condiciones de parada. Generalmente, un AE se compone de un sistema de codificación para los individuos de su población, un algoritmo de inicialización de la población, un método de evaluación de individuos, un conjunto de operadores evolutivos y una condición de parada [MORA08]. La Figura 2.1 muestra cómo se interrelacionan estos componentes. En primer lugar, la población inicial de individuos es generada por el algoritmo de inicialización conforme al sistema de codificación de individuos designado. Seguidamente, la población es insertada en la primera iteración del AE. En cada una de estas iteraciones se evalúa el nivel de adaptación de los individuos de dicha población y se comprueba si ésta cumple los criterios definidos en la condición de parada. En caso afirmativo el algoritmo termina otorgando como resultado la población de individuos. De lo contrario, la población es sometida a los operadores evolutivos y se vuelve a evaluar, continuando con la siguiente iteración del AE.

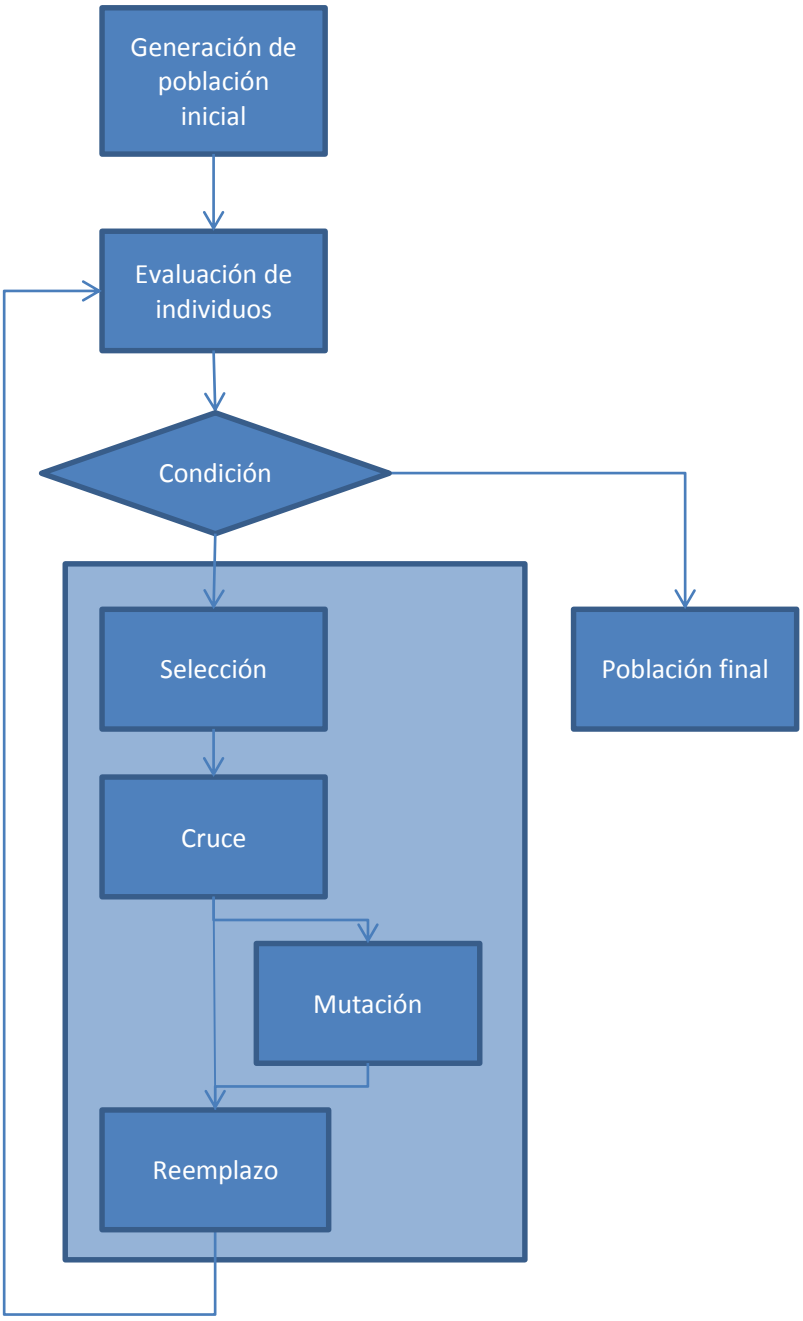


Figura 5 Diagrama de un AG

Cada individuo representa una solución dentro del espacio de búsqueda. De esta forma, la solución final será un individuo obtenido a partir del conjunto de genes dominantes a lo largo del proceso evolutivo.

La descendencia se genera de forma pseudo-aleatoria. Se han desarrollado diferentes tipos de cruce en base al tipo de problema a resolver mediante computación evolutiva. Es posible mantener de una generación a otra el mismo individuo y una copia suya mutada con el fin de no eliminar potenciales buenas soluciones. De la misma forma, la computación evolutiva permite mantener un individuo "inmortal" siempre que esté bien adaptado.

3.1.1 Función de evaluación

La capacidad de adaptación en un AE viene dada por el mecanismo de evaluación de la población. Este mecanismo decodifica el genotipo de cada individuo obteniendo su fenotipo, evaluando su bondad como solución candidata a través de la función de evaluación o función objetivo. La función de evaluación devuelve la medida del grado de adaptación de los individuos a lo largo del proceso evolutivo, y por lo tanto dirige la evolución de la población hacia una determinada región del espacio de soluciones. El resultado de la evaluación es el valor de adaptación del individuo, comúnmente denominado grado de adaptación (*fitness*). Se denomina individuo óptimo u óptimo global a aquel que pertenece al espacio de soluciones y cuyo fenotipo obtiene el mejor grado de adaptación posible como resultado de la función de evaluación. Un óptimo local es todo aquel individuo cuyo grado de adaptación es inferior al grado de adaptación del individuo óptimo pero superior al de los individuos próximos a él en el espacio de soluciones. El grado de adaptación de los individuos es una medida de referencia comúnmente usada durante la evaluación de la condición de parada y la aplicación de los operadores evolutivos. Cuando el óptimo global de un AE es el grado de adaptación más bajo, el algoritmo se enfrenta a un problema de minimización. En caso contrario, el AE se enfrenta a un problema de maximización. Dado que es posible definir una función que transforme un problema de maximización en uno de minimización, se considera que ambos problemas son equivalentes.

3.1.2 Selección de individuos

La selección de individuos está directamente relacionada con el concepto darwiniano de la supervivencia del mejor adaptado. Los progenitores que darán lugar a la siguiente generación son seleccionados entre los individuos de la población en base a su grado de adaptación. Los individuos seleccionados se copian a un lugar intermedio denominado lugar de apareamiento (del inglés *mating pool*). Los individuos más adaptados tienen más probabilidades de ser seleccionados, incluso varias veces si esto

se permite. Por el contrario, los individuos menos adaptados tendrán una probabilidad muy baja de poder producir descendencia. El subconjunto de posibles descendientes a generar por el operador de cruce es definido por la selección de progenitores realizada por el operador de selección, por lo que éste condiciona directamente a la dirección que tomará el proceso evolutivo generación tras generación. Los operadores de selección más comúnmente implementados son los siguientes [FONT09]:

1. Suponiendo una función objetivo no negativa para un problema de maximización del grado de adaptación y siendo $f\left(I_j\right)$ el grado de adaptación del individuo I_j, la probabilidad de que dicho individuo sea seleccionado en una población de *N* individuos se denota como p_j y se obtiene de la siguiente forma:

$$p_j = \frac{f\left(I_j\right)}{\sum_{K=0}^{N-1} f\left(I_K\right)}$$

   La existencia de un individuo muy adaptado en la población puede provocar que éste sea el único individuo seleccionado, debido a que su valor p_j resulta mucho mayor que el del resto de individuos de la población. Este hecho repercute en una baja diversidad genética en el lugar de apareamiento, lo que limita la capacidad de exploración del AE y favorece su caída en un óptimo local.

2. Suponiendo una población de infinitos individuos, el operador de selección por clasificación define la probabilidad p_j de que el individuo I_j ocupe la posición m dentro de dicha población como:

$$p_j = q \cdot (1-q)^{m-1},$$

   distribución geométrica del parámetro q, el cual toma valores en el intervalo $[0, 1]$ y define la probabilidad de que un individuo se encuentre en el primer lugar de la población $(m=1)$. Si bien un AE no trabaja con poblaciones infinitas, la probabilidad de que cualquier individuo ocupe un lugar en una población de tamaño N es

$$\sum_{m=1}^{N} p_j = 1-(1-q)^{N+1},$$

valor próximo a 1 para los valores de q y N usados.

3. El muestreo universal estocástico o método de la ruleta emplea un círculo dividido en tantos sectores circulares como individuos hay en la población, siendo el tamaño de cada sector proporcional al grado de adaptación del individuo al que representa. La selección se realiza trazando una recta con pendiente aleatoria que pasa por el centro del círculo y corta a dos sectores circulares. Los individuos representados en dichos sectores son seleccionados. Se pueden trazar tantas rectas como pares de individuos se desee obtener. La Figura 6 representa un ejemplo en el que el individuo I_1 (el más adaptado) se escoge dos veces, los individuos I_2 e I_3 se escogen una vez cada uno, mientras que el individuo I_4 (el menos adaptado) no es seleccionado.

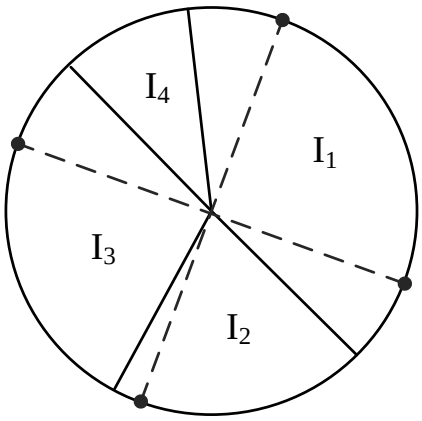


Figura 6 Operador de selección universal estocástico o de la ruleta

4. La selección por torneo construye tantos subconjuntos de individuos extraídos al azar de la población como progenitores se desean seleccionar. El tamaño de todos los subconjuntos es el mismo, siendo menor que el tamaño de la población. Cada subconjunto constituye un torneo cuyo vencedor es el individuo con mejor grado de adaptación. Esta condición de victoria puede sustituirse por una medida probabilística similar a la utilizada en la selección por clasificación.

3.1.3 Operadores de reemplazo

El concepto de reemplazo suele ir ligado con el de tasa de reemplazo generacional $t_{tg} \in (0,1]$, que indica el porcentaje de descendientes obtenidos en cada generación con respecto al tamaño de la población N. Inicialmente, el reemplazo se efectuaba de uno en uno, $t_{tg} = N^{-1}$, por lo que se generaba un único individuo descendiente que sustituía a otro de la población [HOLL92]. Es posible realizar reemplazos generacionales completos, de tal forma que $t_{tg} = 1$. Los operadores de reemplazo más extendidos evitan estos extremos y usan una tasa de reemplazo intermedia, $t_{tg} \in (0,1)$, que únicamente sustituye una parte de los individuos de la población. Este tipo de algoritmos genéticos son denominados SSGA, por las siglas del término inglés Steady-State Replacement Genetic Algorithms.

Al igual que la selección de individuos, el reemplazo suele realizarse atendiendo al grado de adaptación de cada uno de los individuos de la población. Las mecánicas de reemplazo más comunes son las siguientes [ENGE07] (Engelbrecht, 2007):

1. El reemplazo de los peores individuos sustituye a los individuos peor adaptados de la población por los descendientes generados.

2. El reemplazo aleatorio de los individuos de la población opera igual que el anterior pero elimina a los individuos aleatoriamente.

3. El reemplazo por torneo actúa de una manera similar al operador de selección que lleva el mismo nombre. Se establecen tantos torneos, de un tamaño menor que la población, como individuos se requiera eliminar. El ganador de cada torneo es el individuo peor adaptado y resulta eliminado de la población.

4. El reemplazo de los más viejos elimina a aquellos individuos que hayan permanecido más generaciones en la población de individuos.

Existen estrategias de reemplazo que modifican en cierta medida la aplicación de estos operadores. Una estrategia elitista exime al individuo mejor adaptado de ser reemplazado, asegurando su permanencia en la siguiente generación. Una estrategia de reemplazo de padres puede provocar, con una cierta probabilidad, que la descendencia ocupe el lugar de sus progenitores en la población, independientemente del grado de adaptación de los mismos. Adicionalmente se encuentra un grupo de operadores relacionados con los métodos de reemplazo de individuos, denominados catastróficos (Font et al., 2009). Dichos operadores realizan una evaluación cada cierto

tiempo sobre el grado de convergencia del algoritmo para decidir si reemplazar una serie de individuos de la población actual por otros generados aleatoriamente. Estos operadores aumentan la diversidad genética de la población preservando a los mejores individuos generados hasta el momento. Esta diversidad genética potencia la capacidad de exploración del AE en detrimento de su capacidad de explotación. Los principales operadores catastróficos son dos:

1. El operador de empaquetado elimina a todos los individuos que tienen el mismo grado de adaptación excepto a uno, que permanece como representante de dicho valor de adaptación. El resto son reemplazados en la población por individuos generados aleatoriamente.

2. El día del juicio final destruye a todos los individuos de la población excepto al más adaptado. La población se vuelve a rellenar con individuos generados aleatoriamente.

3.1.4 Condiciones de parada

El flujo de ejecución de un AE es cíclico y cada iteración que se ejecuta sobre él recibe el nombre de generación. El AE converge cuando alcanza un estado estable en el que la población de individuos está estancada, es decir, que no puede progresar aunque se sigan sucediendo las generaciones. Un AE puede converger en un óptimo global o local. En el primer caso, la población converge habiendo encontrado un individuo que obtiene de la función objetivo el mejor grado de adaptación posible. En el segundo caso, el AE ha quedado encallado en una región del espacio de soluciones fuera de la cual no ha sido capaz de explorar. Cuando un AE converge en un óptimo local, ninguno de los individuos de su población obtiene el mejor grado de adaptación de la función objetivo. Cuando el problema a resolver carece de una solución de valor óptimo conocido, es difícil discernir si el AE ha convergido prematuramente o ha alcanzado un óptimo global, por lo que se establece un criterio, denominado condición de parada, que determina cuándo debe finalizar su ejecución. Algunas condiciones de parada toman como referencia el grado de adaptación de los individuos de la población, como la no progresión del valor de mejor grado de adaptación durante un determinado número de generaciones o cuando un alto porcentaje de los individuos comparte el mismo grado de adaptación. Otras condiciones de parada atienden a criterios tales como la proximidad en el genotipo de los individuos o el alcance de un número máximo preestablecido de generaciones.

3.2 Algoritmos genéticos

Los algoritmos genéticos son una clase de algoritmos evolutivos empleados en problemas de optimización y búsqueda que se caracterizan por codificar los individuos como vectores de dimensiones fijas. Los algoritmos genéticos utilizan los procesos de la evolución descritos en el apartado anterior para obtener individuos o soluciones al problema cada vez más adaptados. Las claves del éxito de un algoritmo genético están en su codificación y los métodos de cruce y selección empleados.

3.2.1Sistema de codificación

La codificación del problema es el que establece la representación de una solución al problema como un conjunto de valores. La codificación imita el sistema de cromosomas y genes de los seres vivos. En los algoritmos genéticos habitualmente se utiliza un único cromosoma que es el que contiene toda la cadena de genes que codifican la solución. Cada posible valor que pueda tomar un gen se denomina alelo.

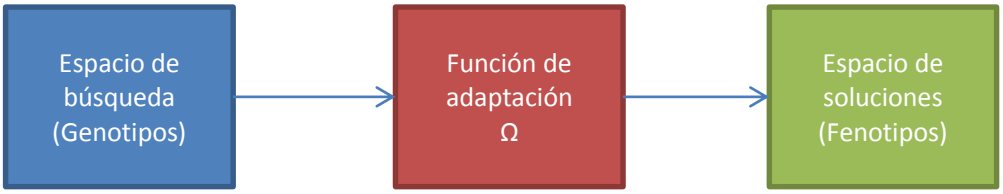


Figura 7 Sistema de codificación de un AG

Habitualmente, la codificación utilizada suele ser completa. Es decir, si se utiliza un alfabeto A de cardinal finito m, y cadenas de longitud l, cualquiera de las m^l posibles cadenas corresponde a la codificación de un único punto del dominio de la función Ω [BARR91]. Este tipo de codificación discretiza la solución, por lo que es posible que no exista una relación biunívoca entre el dominio de la función Ω y el conjunto de posibles soluciones permitidas por la codificación. Una buena codificación es la que maximiza tanto el espacio de búsqueda como el espacio de soluciones obtenidas mediante la función Ω. Una mala elección en la codificación ralentiza sustancialmente la convergencia del algoritmo.

3.2.2 Operadores de cruce

Los operadores de cruce en los algoritmos genéticos dependen del tipo de codificación elegida, pudiendo ser de alfabeto finita o de alfabeto infinito.

Suponiendo que para codificar los individuos de un AG se emplea un alfabeto finito A de cardinal m, la cadena $c_1, c_2, ..., c_l$ corresponde a la codificación de un punto del espacio de búsqueda, donde $c_k \in A$, para cada $k \in \{1,2,...,l\}$, utilizando cadenas de longitud *l*. Algunos de los operadores de cruce más extendidos aplicables a esta codificación son los siguientes [MANR01]:

1. El operador de cruce basado en un punto toma dos padres y corta sus genotipos en un punto elegido al azar, denominado lugar de cruce, obteniendo dos cadenas de código genético de cada progenitor. La primera cadena resultante del primer progenitor se combina con la segunda resultante del segundo progenitor para dar lugar al genotipo de un primer descendiente. El genotipo del segundo descendiente se compone de la primera cadena del segundo progenitor y la segunda cadena del primer progenitor. De esta forma se crean dos descendientes que heredan genes de cada uno de los padres.

2. El operador de cruce multipunto es una extensión del anterior en el que se establecen n lugares de cruce, dando lugar a $n+1$ cadenas resultantes de cada uno de los progenitores. El genotipo de los dos descendientes se obtiene recombinando de manera alterna dichas cadenas, análogamente al procedimiento seguido por el operador basado en un punto. Se han realizado diferentes investigaciones en cuanto al comportamiento del operador de cruce basado en múltiples puntos, concluyéndose que el basado en dos puntos representa una mejora respecto al operador basado en uno solo. Sin embargo, el añadir más puntos de cruce no beneficia al comportamiento del algoritmo.

3. El operador de cruce uniforme engloba a los dos anteriores operadores. Se basa en la definición de una máscara de cruce consistente en una cadena de bits aleatoria de longitud l. Dados dos genotipos progenitores $a_1, a_2, ..., a_i, ..., a_l$ y $b_1, b_2, ...,b_i, ..., b_l$, la posición i de la cadena del primer descendiente será a_i si existe el valor 0 en la posición i de la máscara de cruce; y será b_i en caso contrario. Para el caso del segundo descendiente se aplica la máscara de forma inversa. Nótese que si la máscara de cruce consta de *k* ceros seguidos de $l-k$ unos, el resultado del operador de cruce uniforme es idéntico al de la aplicación de un operador de cruce en un solo punto en la posición k.

4. Sea $S = \{0,1\}^l$ el conjunto de todas las posibles cadenas binarias de longitud l y $g : S \to \{0,1,\ldots,2^l - 1\}$ la función biyectiva que permite decodificar una cadena binaria en un número entero. Dados dos genotipos progenitores $a,b \in S$, el operador de cruce generalizado (Barrios, 1991) define el intervalo de cruce generado por ambos como $[g(a \vee b), g(a \wedge b)]$, siendo $\wedge$ el operador lógico *and* bit a bit y $\vee$ el operador lógico *or* bit a bit. Los descendientes generados por este operador son los puntos de este intervalo a' y b' tales que:

$$a' \in g^{-1}\left(\left[g(a \wedge b), g(a \vee b)\right]\right) ; b' = g^{-1}\left(g(a) + g(b) - g(a')\right)$$

La codificación más usual para un AG con alfabeto infinito es el alfabeto de números reales. Partiendo de una codificación con alfabeto real para un AG, se obtienen dos progenitores cuyos genotipos son las cadenas de números reales $a = a_1, a_2, ..., a_l$ y $b = b_1, b_2, ..., b_l$ con $a_i, b_i \in \mathbb{R}, \forall i \in \{1, ..., l\}$. Los operadores de cruce más comúnmente aplicados a esta codificación son [MANR01]:

1. El operador de cruce uniforme puede aplicarse a una codificación real de manera análoga a su aplicación sobre una codificación binaria. El inconveniente de este operador es que únicamente recombina los genes de los genotipos de los progenitores, por lo que los genes de los descendientes jamás tomarán alelos distintos de los que toman los genes de los progenitores.

2. El cruce aritmético genera dos individuos $o^1 = o_1^1, o_2^1, \ldots, o_l^1$ y $o^2 = o_1^2, o_2^2, \ldots, o_l^2$ con $o_i^1, o_i^2 \in \mathbb{R}, \forall i \in \{1,...,l\}$, a partir de una media ponderada de los genotipos de los progenitores. El valor del gen o_i^1 resulta de la expresión $o_i^1 = r \cdot a_i + (1-r) \cdot b_i$, donde $r \in [0,1]$ es un número aleatorio. Análogamente, el valor o_i^2 se obtiene mediante la expresión $o_i^2 = r \cdot b_i + (1-r) \cdot a_i$, para el mismo valor de r.

3. El operador de cruce plano puede generar n individuos a partir del intervalo que generan los genotipos de dos progenitores. Para cada gen o_i de un descendiente, se selecciona aleatoriamente un alelo dentro del intervalo

$\left[max\{a_i,b_i\},min\{a_i,b_i\}\right]$, definido por los alelos de los genes de los dos progenitores en la posición i.

4. El cruce BLX-α calcula n genotipos descendientes de la forma $o^n = o_1^n, o_2^n, \ldots, o_l^n$ con $o_i^n \in \square, \forall i \in \{1,...,l\}$, a partir de la ampliación del intervalo generado por los genotipos progenitores. De esta forma, el alelo del gen o_i^n es generado aleatoria e uniformemente en el intervalo $\left[min(a_i,b_i) - I \cdot \alpha,\ max(a_i,b_i) + I \cdot \alpha\right]$, donde $\alpha \in [0,1]$ y el valor de I se calcula como $I = max\{a_i,b_i\} - min\{a_i,b_i\}$. El cruce plano es un caso particular de este operador para $\alpha = 0$.

3.2.3 Operaciones de mutación

El operador de mutación por intercambio establece dos lugares de mutación en un genotipo seleccionando al azar dos de sus genes. La mutación ocurre intercambiando estos genes de posición. Este operador de mutación es sencillo y aplicable a cualquier tipo de codificación. Sin embargo, no es capaz de hacer que los genes seleccionados muten a un alelo que no está presente en el genotipo del individuo. El operador de mutación uniforme solventa esta limitación seleccionando únicamente un lugar de mutación y sustituyendo aleatoriamente su valor por cualquier alelo del alfabeto de símbolos de la codificación. Cuando se aplica a alfabetos binarios, el efecto de este operador se limita a cambiar un alelo '0' por un '1', o viceversa. El operador de mutación no uniforme, únicamente aplicable a codificaciones reales con alfabeto $[LB,UB]$, selecciona al azar el gen a_i y lo sustituye por el gen a_i', cuyo valor se define de la siguiente forma:

$$a_i' = \begin{cases} a_i + \Delta(t, UB - a_i), & si\ v = 1 \\ a_i - \Delta(t, a_i - LB), & si\ v = 0 \end{cases},$$

donde $\Delta(t,y)$ es una función que devuelve un valor en el intervalo $[0, y]$, t el valor de la generación actual del algoritmo, y v un valor booleano aleatorio. Su objetivo es variar el alelo de un gen dentro de un intervalo centrado en su valor actual, siendo este intervalo más pequeño a medida que las generaciones del AG avanzan [MORA08].

3.3 Programación Genética Guiada por Gramáticas

La Programación Genética es un tipo de algoritmo evolutivo donde los individuos se codifican como programas de longitud variable, en contraposición a los AAGG, donde se codifican como vectores de dimensión fija. La población de un PG se compone de un conjunto de programas que evolucionan con el objetivo de encontrar aquel que representa una solución adecuada al problema que se desea resolver.

La programación genética busca construir programas de computación sin ningún tipo de diseño previo, únicamente empleando procesos de la computación evolutiva. En este caso, se escogen como elementos que conforman los individuos las acciones de los programas (funciones) y los datos sobre los que actúan los programas (terminales), y se opera con ellos hasta encontrar una estructura que represente una solución óptima al problema [KOZA92].La programación genética guiada por gramáticas es una extensión de los sistemas tradicionales de PG que utiliza gramáticas libres de contexto (CFG, del inglés Context-Free Grammar) para establecer las definiciones formales y las restricciones sintácticas del problema a resolver. Un Programa Genético Guiado por Gramáticas (PGGG) es capaz de encontrar soluciones a cualquier problema cuyas restricciones puedan ser codificadas en las reglas de producción de una CFG. Esta gramática genera el lenguaje cuyas palabras son todas las posibles soluciones a dicho problema, por lo que define el espacio de soluciones del PGGG. La población de individuos es un subconjunto del espacio de soluciones cuyos genotipos son árboles de derivación de la CFG. Este formalismo garantiza que todos los individuos generados por un PGGG son gramaticalmente válidos y que, por tanto, codifican soluciones formalmente correctas para resolver el problema objetivo.

3.3.1 Codificación de individuos y generación de la población inicial

La generación de la población inicial de individuos de un PGGG tiene como fin la construcción aleatoria de un número determinado de individuos que pertenezcan al lenguaje generado por una CFG. El algoritmo de fuerza bruta [KARP87], propuesto por Richard M. Karp como un algoritmo de comparación de patrones ampliable y de propósito general, es fácilmente adaptable para la generación automática de una población de árboles de derivación. Dada la CFG, el algoritmo escoge aleatoriamente entre las producciones que conforman la gramática para ir formando cada uno de los

individuos que componen la población. Los individuos generados son válidos sintácticamente, puesto que se generan siguiendo las producciones de la gramática. Para evitar la generación de árboles con profundidad desmesurada, se establece un parámetro que define la profundidad máxima permitida para los individuos de la población. Sin embargo, este método sólo es capaz de controlar la profundidad de los árboles de derivación una vez han sido generados, por lo que debe descartar todos aquellos individuos cuya profundidad exceda de la máxima permitida. Esto da lugar a un proceso ineficiente en términos de coste computacional.

El algoritmo de generación aleatoria de individuos basado en gramáticas (García-Arnau et al., 2007) propone un método de generación aleatoria de la población con control de la profundidad. La elección de las producciones que generan un genotipo no se realiza completamente al azar, sino comprobando a cada paso tanto su validez gramatical como su nivel de profundidad. Esto proporciona un ahorro computacional importante en términos de tiempo y memoria, dado que ningún individuo válido es descartado y ningún individuo no válido es generado para formar parte de la población inicial. El algoritmo recibe dos parámetros de entrada: el número de individuos (*N*) que componen la población y la profundidad máxima permitida (*D*), sin contar la raíz, de los árboles de derivación. El proceso de generación aleatoria de la población consta de tres pasos:

1. Se anota la longitud para cada producción de la gramática. Para ello se tienen en cuenta las siguientes reglas:
    a. La longitud de un símbolo no terminal A es el mínimo de las longitudes de todas sus producciones, y se denota por $L(A)$.
    b. La longitud de un símbolo terminal a es 0 porque no deriva en nada, denotándose por $L(a) = 0$.
    c. La longitud de una producción $A ::= \alpha$ es el resultado de sumar uno al máximo de las longitudes de los símbolos que componen la parte derecha, y se denota por $L(A ::= \alpha)$.
    d. Las producciones que generan sólo símbolos terminales tienen longitud 1, denotándose por $L(A := a) = 1$.

2. Se calcula la longitud del axioma como el mínimo de las longitudes de todas aquellas producciones cuyo antecedente es el axioma. Esta longitud determina la profundidad mínima permitida (d) para un individuo válido de esa gramática.

3. Repetir N veces:
    a. Se escoge aleatoriamente un valor comprendido entre la profundidad mínima permitida (*d*) y la profundidad máxima permitida (*D*). Este valor corresponde a la profundidad máxima elegida (*p*) para el individuo que se está generando. El algoritmo garantiza la generación de un individuo cuya profundidad está comprendida entre d y p.
    b. Se etiqueta el axioma como el símbolo no terminal actual A y se asigna el valor 0 a la profundidad actual (PA).
    c. Se selecciona aleatoriamente una producción de la forma $A ::= \alpha$, que cumpla que $PA + L(A ::= \alpha) \leq p$.
    d. Por cada no terminal $B \in \alpha$, se anota B como el símbolo no terminal actual y se repiten los pasos 3.c y 3.d, incrementando en una unidad el valor de PA. El algoritmo finaliza cuando todos los nodos hoja del árbol contienen símbolos terminales.

Producción	Longitud
S ::= EXP = EXP	3
EXP ::= (*EXP OPR EXP*)	3
EXP ::= (*EXP* + *EXP*)	3
EXP ::= (*EXP* - *EXP*)	3
EXP ::= NUM	2
OPR ::= + | -	1
NUM ::= 0 | 1 | 2 | 3 | … | 9	1

Tabla 1 Longitud de las producciones de la gramática GEXP

La Tabla 1 muestra las producciones de la gramática G_{EXP} con sus correspondientes longitudes. Dado que la longitud del axioma es 3, la profundidad mínima permitida para los individuos generados a partir de esta gramática es $d = 3$. La Figura 8 muestra la generación paso a paso del individuo. En cada nivel, todo nodo que contiene un símbolo no terminal elige una de sus producciones, cumpliendo que la longitud de la misma sumada a la profundidad actual no sobrepase el valor de p. Cuando todos

los nodos no terminales del nivel actual han elegido una producción válida, se desciende al siguiente nivel y repite el procedimiento. En el nivel de $PA=3$ solo se eligen producciones que derivan en nodos terminales, por lo que el árbol queda completamente desarrollado con una profundidad de 4.

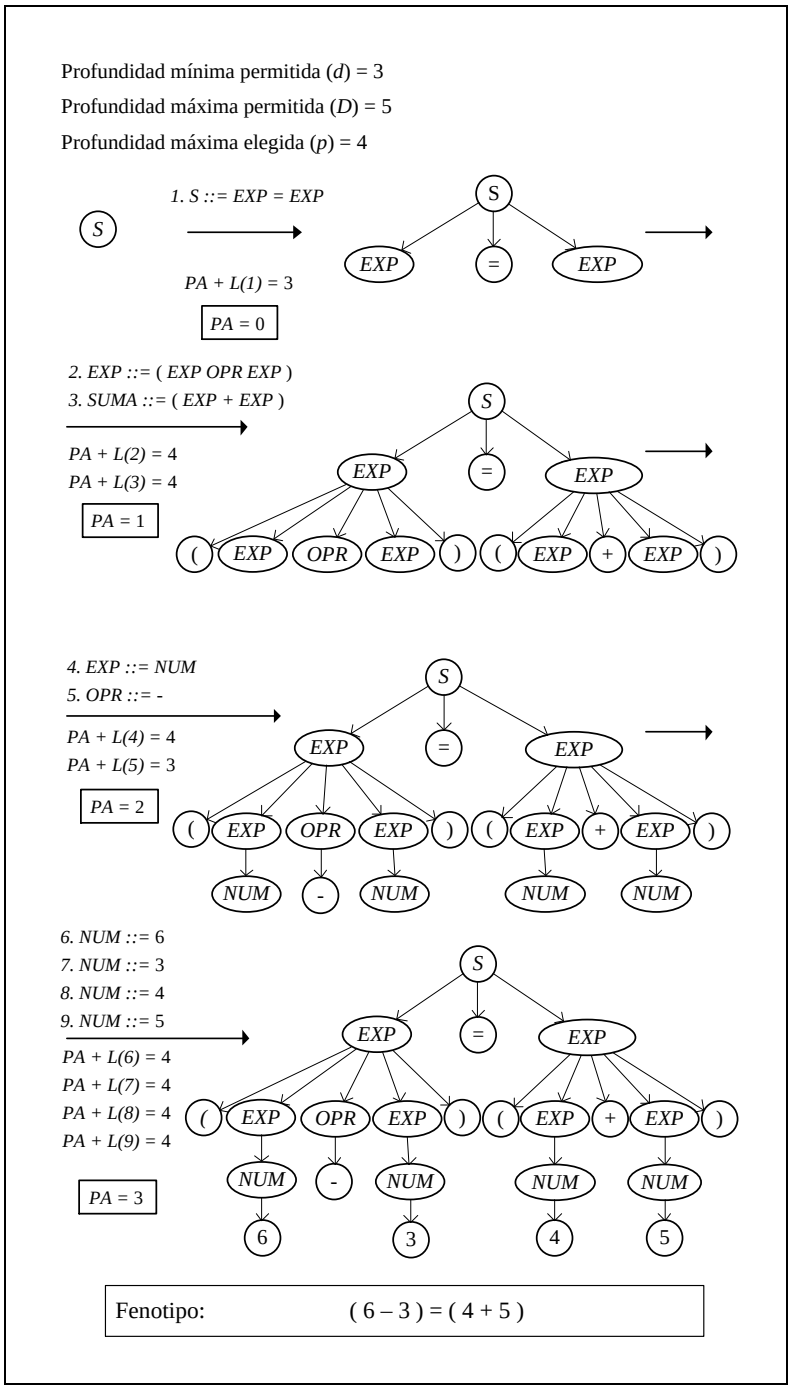


Figura 8 Generación del genotipo que codifica el fenotipo " (6 - 3) = (4 + 5) ", perteneciente al lenguaje desarrollado por la gramática GEXP

3.3.2 Operadores de cruce

El cruce se realiza mediante la selección de los nodos de los padres que se van a intercambiar. Una vez elegidos los nodos, se intercambia todo el subárbol desde cada uno de los nodos. La elección de un correcto operador de cruce es fundamental por diversas causas. La selección de nodos al azar puede provocar que se generen individuos no válidos al finalizar el proceso. También puede surgir la llamada explosión de código [WHI95], que es la generación desproporcionada de código no útil o poco eficiente.

El operador propuesto por Whigham es comúnmente utilizado en PGGG y consta de los siguientes pasos:

1. Se escoge un nodo de manera aleatoria en el primer progenitor y se le denomina nodo de cruce. La elección de este nodo se restringe al conjunto de todos aquellos nodos que contengan símbolos no terminales.

2. Se escoge un nodo no terminal que pertenezca al segundo progenitor, de entre todos aquellos nodos de su genotipo que albergan el símbolo no terminal del nodo elegido en el primer progenitor. De esta forma se asegura la generación de individuos gramaticalmente válidos.

3. Se intercambian los subárboles cuyas raíces son los nodos elegidos de ambos progenitores. Los árboles de derivación resultantes son los genotipos gramaticalmente válidos de la descendencia generada.

La Figura 9 muestra el cruce de dos individuos pertenecientes al lenguaje generado por la gramática G_{EXP}, cuyos genotipos codifican los fenotipos "$(6 - 3) = (4 + 5)$" y "$8 = (7 + 7)$", respectivamente. El genotipo del primer progenitor muestra en color gris el nodo de cruce elegido aleatoriamente en él, que contiene el símbolo no terminal "*EXP*". El genotipo del segundo progenitor muestra en gris el conjunto de todos los nodos susceptibles de ser escogidos como segundo nodo de cruce. La línea circular discontinua marca el segundo nodo de cruce seleccionado al azar de dicho conjunto. Los genotipos de los descendientes mostrados en la parte inferior de la figura resultan del intercambio de los subárboles cuyas raíces son los nodos de cruce seleccionados. Estos descendientes codifican los fenotipos "$((7+7)-3)=(4+5)$" y "$8 = 6$", respectivamente, y son gramaticalmente válidos.

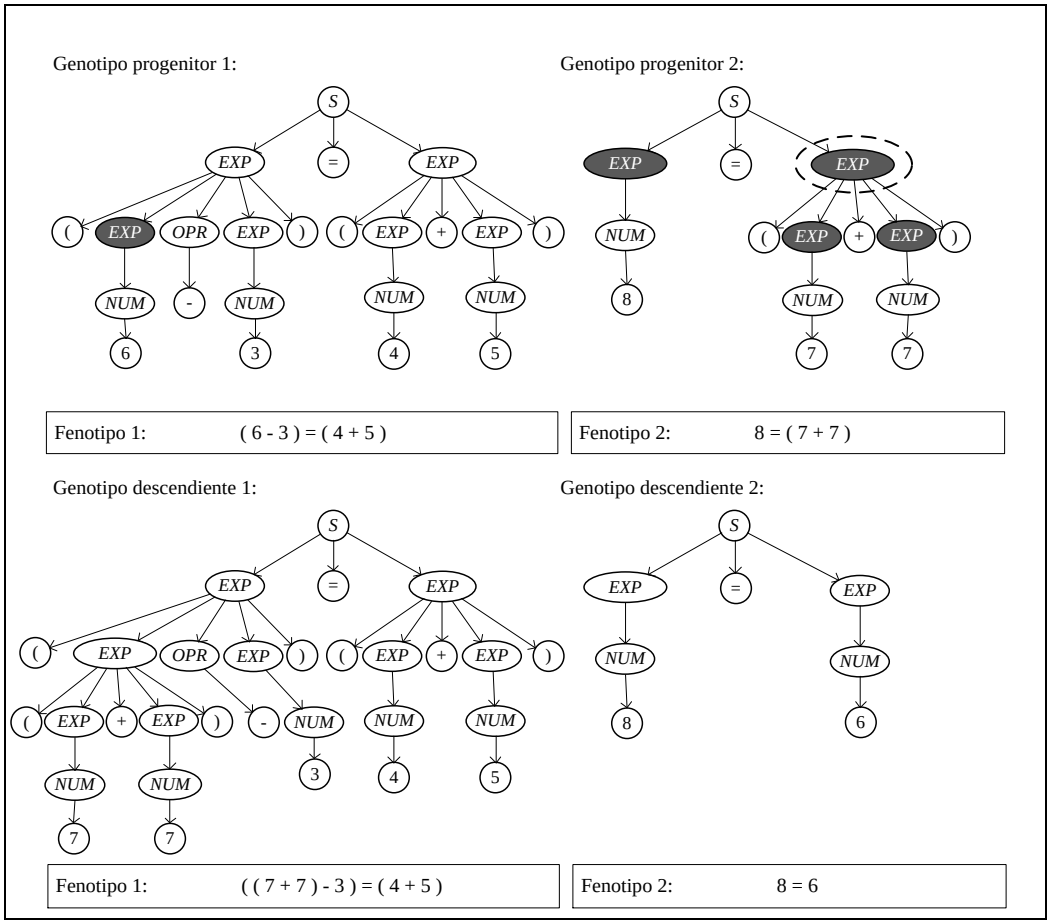
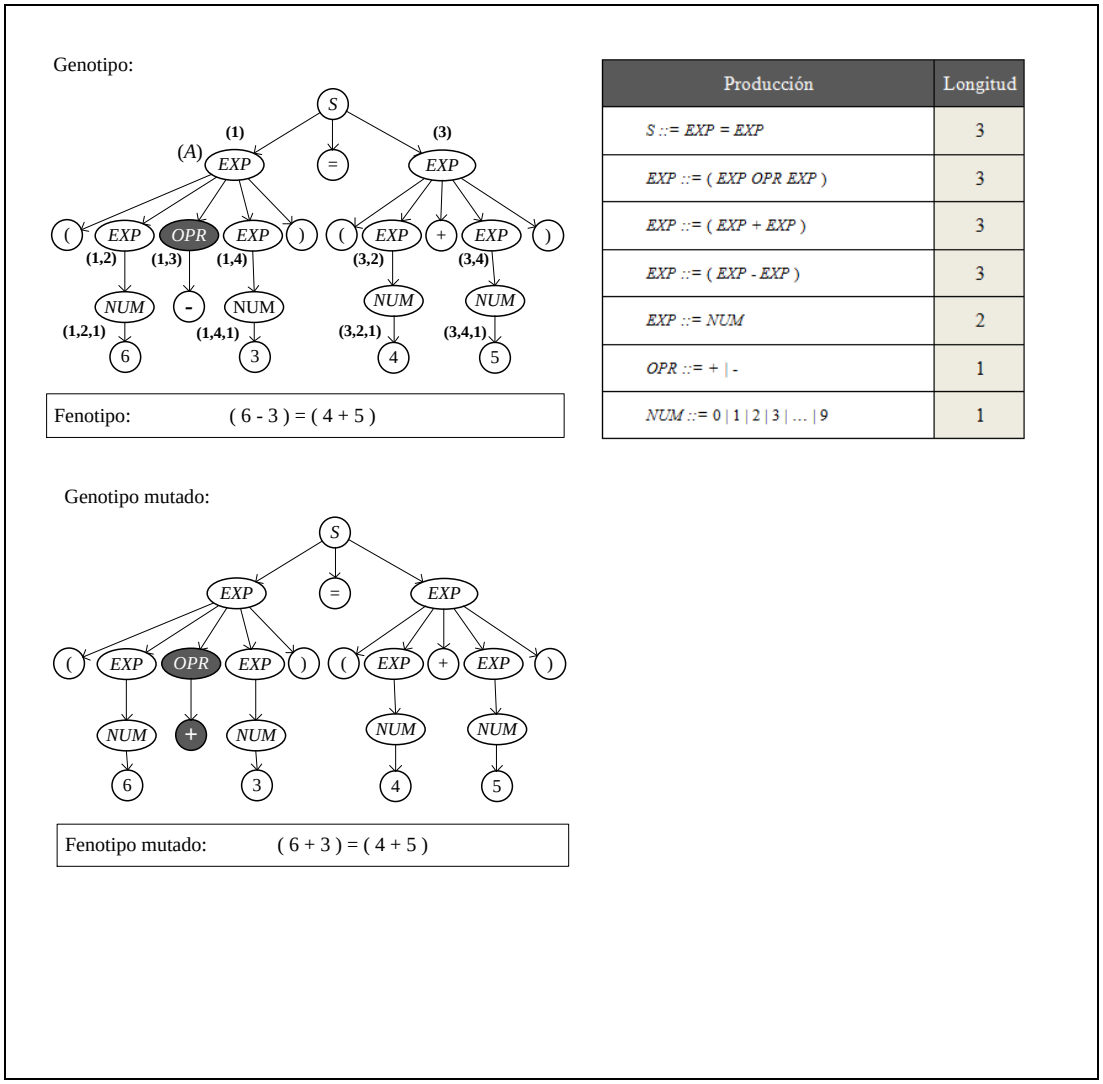


Figura 9 Cruce de individuos mediante el operador de cruce de Whigham

El principal inconveniente del operador de cruce de Whigham es que no obtiene ventaja de las gramáticas ambiguas, como G_{EXP}. Una gramática ambigua es aquella que permite generar individuos con diferentes genotipos que codifican el mismo fenotipo. Para un mismo problema, el leguaje generado por una gramática ambigua provee un mayor número de soluciones óptimas que el generado por una gramática no ambigua.

3.3.3 Operadores de mutación

El operador de mutación permite introducir un nuevo subárbol dentro de la codificación de un individuo sustituyéndolo por otro nodo ya existente. El problema de esta definición está en que en la mayoría de los casos el individuo generado no es válido debido a que el nodo insertado no suele ser un hijo válido del padre al que se le ha asociado. Para solventar este problema, Whigham propone una mutación donde se elige un nodo y se sustituye todo su subárbol por otro cuya raíz es del mismo tipo que el nodo elegido y con una longitud tal que el individuo resultante no exceda de la profundidad máxima definida. De esta forma se garantiza que todos los individuos generados mediante mutación son válidos.



Producción	Longitud
S ::= *EXP* = *EXP*	3
EXP ::= (*EXP OPR EXP*)	3
EXP ::= (*EXP* + *EXP*)	3
EXP ::= (*EXP* - *EXP*)	3
EXP ::= *NUM*	2
OPR ::= + | -	1
NUM ::= 0 | 1 | 2 | 3 | … | 9	1

Figura 10 Mutación de un individuo mediante el operador de mutación Whigham

4 Inteligencia artificial en videojuegos

Se define inteligencia artificial (IA), desde el punto de vista académico, como programas de ordenador que emulan acciones y pensamientos humanos, así como acciones y pensamientos racionales [RUSS95]. La definición engloba los enfoques cognitivo y de comportamiento de la inteligencia. Incluye también las nociones de racionalidad y humanidad. En los videojuegos, se considera inteligencia artificial al código de un juego que permite a los elementos controlados por el ordenador simular la toma de decisiones inteligentes cuando existen diversas opciones para una situación dada [SCHW09]. Por lo tanto, la IA de los juegos está enfocada a los comportamientos y, especialmente, a sus resultados más que a la forma en la que se llega a estos. Es decir, lo más importante en la IA de los videojuegos es qué hace, no cómo ha decidido hacerlo.

Es habitual que las personas ajenas a la IA intenten entender cómo una máquina o un robot piensan. Por ejemplo, cuando un robot decide girar en una dirección para evitar un obstáculo, se suelen dar motivos a ese comportamiento tales como "sabía que iba a chocar y ha dado la vuelta" o "va a explorar otras zonas porque ahí no hay nada". Normalmente un robot gira debido a que un sensor de proximidad emite una señal que hace que el robot cambie de dirección. En el mundo de los juegos, la diversión se logra haciendo que parezca que un elemento del juego es inteligente, independientemente de los métodos que se empleen para que lo aparente.

4.1 Técnicas de inteligencia artificial en videojuegos de lucha

En la década de los 90, los juegos de lucha fueron el género más extendido en el mundo de los videojuegos [USGM13]. Este género se caracteriza por presentarse en un escenario estático donde dos luchadores ejecutan movimientos para reducir la vida del oponente en un tiempo determinado. El auge de estos juegos vino principalmente por introducir un elemento novedoso, los "combos" o combinaciones de botones siguiendo una secuencia determinada para que el personaje realizase movimientos complejos y muy vistosos. Esta técnica popularizó títulos como *Street Fighter 2: The World Warrior* de Capcom [USGM13].

En este género hay 2 jugadores simultáneos por lo que es necesario un sistema inteligente que pueda controlar a un jugador, mientras que un humano maneja al otro. Las principales técnicas empleadas en estos juegos para el desarrollo de sistemas

inteligentes son 3: las máquinas de estados finitas (*FSM* del inglés *Finite State Machines*), los sistemas guiados por datos y la programación guiada (*scripting* en inglés) [SCHW09].

4.1.1 Máquinas de estados finitas

Las máquinas de estados finitas definen un conjunto de situaciones o estados en los que un personaje puede estar. A cada estado se le asigna una o varias acciones que el personaje puede realizar. Para añadir diversión al juego, las acciones se seleccionan de forma aleatoria de entre todas las posibles. La Figura 11 muestra un ejemplo de una *FSM*. En este ejemplo, un luchador que comience en el estado "de pie", puede ejecutar un movimiento de "agacharse" que provoca que pase a "agachado". Estando en este estado, puede volver a ejecutar otra acción que lo devuelva a "de pie". No todas las transiciones se deben a acciones realizadas por el personaje. Estando en el estado "en el aire", se pasa al estado "de pie" por la propia mecánica del juego, no por acción directa del jugador.

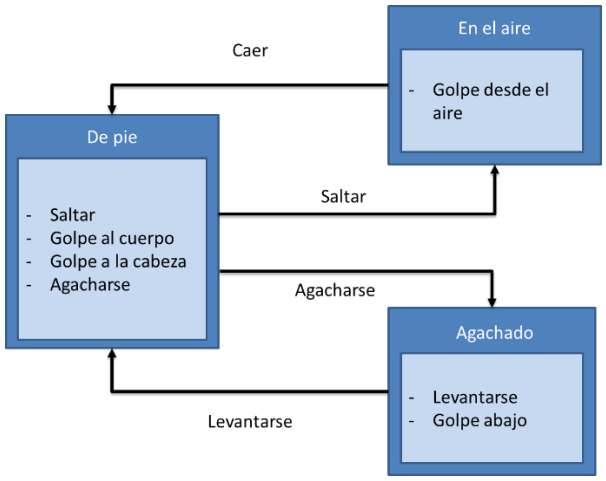


Figura 11 Ejemplo de máquina de estados finita en un juego de lucha

Este tipo de técnicas permiten crear autómatas de forma intuitiva y permite al desarrollador añadir nueva funcionalidad sin aumentar excesivamente la complejidad del sistema.

Los inconvenientes de esta técnica aparecen cuando crece el número de estados. Cada estado nuevo implica comprobar las transiciones del resto de estados hacia el nuevo y viceversa. Además, es habitual que cada personaje de un juego tenga sus propios

movimientos. Esto implica que en muchas ocasiones las máquinas de estados no puedan ser reutilizables con diferentes personajes.

4.1.2 Sistemas guiados por datos

Esta técnica se basa en los sistemas de ayuda a la decisión, donde, dados unos datos de entrada y conociendo el sistema, se ejecuta la acción más conveniente para dicha entrada [POWE02]. En los juegos de lucha, se emplea como conjunto de datos de entrada las acciones del oponente y, conociendo perfectamente la mecánica del juego, se aplica la acción más beneficiosa para el jugador.

Los juegos deben estar equilibrados para que resulten entretenidos. Si siempre hubiese una táctica ganadora, el juego no sería interesante. En la mayoría de los videojuegos se emplea el denominado juego de suma-cero, de la teoría de juegos [MEYR91]. Se trata de una representación matemática donde las victorias por la habilidad de un participante están perfectamente balanceadas con las derrotas por las habilidades de otro participante. El mejor ejemplo de esta definición es el juego piedra-papel-tijeras. Cada una de las habilidades de los participantes (piedra, papel y tijeras) gana a uno pero pierde con otro (Figura 12).

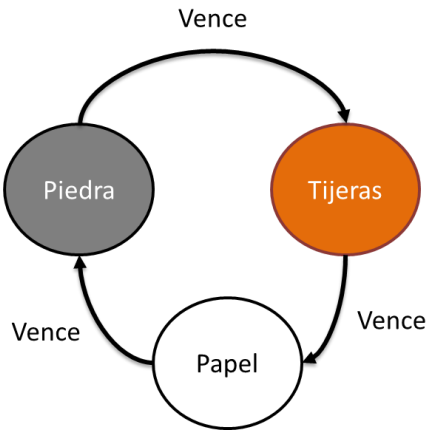


Figura 12 Ejemplo de juego de suma-cero

En el juego de lucha de la competición *Fighting Game AI Competition* [ICEC15] se emplea una tabla que relaciona los ataques realizados, con las defensas que permiten bloquearlos (Tabla 2).

- Guard type-Attack type Conversion Table -

Guard type / Attack type	Stand guard	Crouch guard	Air guard
High	block	block	block
Middle	block	hit	block
Low	hit	block	hit
Throw	hit	hit	miss

Tabla 2 Relaciones de ataque-defensa

Al estar perfectamente definidas las relaciones entre acciones, el factor determinante en este tipo de sistemas inteligentes es la rapidez a la que se ejecuta cada acción. El problema de estos sistemas está en la complejidad de su desarrollo. Cada personaje de un juego de lucha tiene sus propias características. El número de combinaciones de acciones de cada personaje con todos los demás puede ser tan alta que en la práctica no sea viable implementar todas las posibles reacciones.

4.1.3 Programación Guiada

La programación guiada (o *scripting*) consiste en ejecutar un guion o una serie de secuencias de acciones previamente definidas durante el desarrollo del juego [SCHW09]. Esta técnica suele ser utilizada para realizar animaciones o secuencias de vídeo en un juego. Sin embargo, también se emplea para programar los oponentes virtuales en los títulos de lucha. Como indica su definición, el scripting consiste en programar un conjunto de acciones que pueden basarse o no en ciertas entradas al sistema. Esta técnica es la más alejada de la inteligencia artificial, al no tratar de imitar un comportamiento inteligente o algún sistema de pensamiento.

Las ventajas del *scripting* están en la agilidad del desarrollo inicial. Unas pocas instrucciones pueden dar la sensación de un comportamiento inteligente. En los videojuegos suele ser suficiente para crear una buena experiencia de juego. El problema de esta técnica reside en su dificultad a la hora de depurar el sistema. El desarrollo de un sistema inteligente mediante *scripting* es muy tedioso y difícil de modificar.

4.2 Lógica difusa aplicada en videojuegos

La lógica difusa fue introducida en el mundo de los videojuegos por primera vez en 1996 [OBRI96]. Desde entonces se ha aplicado en multitud de juegos y existen múltiples libros de programación orientada a videojuegos que dedican varios capítulos a esta técnica [BOUR04] [HAIG09]. La lógica difusa se emplea, principalmente, combinada con otras técnicas, como las máquinas de estado finitas o las redes neuronales.Se suele utilizar en la toma de decisiones de los *NPC* (Non Playeable Character, personaje no jugable controlado por el ordenador) a la hora de seleccionar qué armas llevar, dónde dirigirse o identificar la amenaza que suponen otros jugadores [FONT12].

Una de las principales ventajas de la lógica difusa está en la facilidad para el desarrollador de los juegos. Los conocimientos previos que necesita esta técnica son la lógica proposicional y el lenguaje natural, los cuales conocen todos los desarrolladores de videojuegos. El uso de las etiquetas lingüísticas permite a los expertos en el dominio del juego plasmar su conocimiento de forma más intuitiva que usando otras técnicas. Uno de los géneros más populares a día de hoy es el disparos en primera persona (*firts person shooter, FPS* en inglés) [ACTI09]. En este tipo de juegos es habitual emplear tácticas militares para moverse por el escenario. Un NPC debe ser capaz de coordinarse con otros NPCs y, en ocasiones, con otros jugadores humanos. Un militar profesional puede exponer su conocimiento y mejorar la calidad de un juego gracias a la naturalidad que le aporta la lógica difusa, empleando únicamente sentencias condicionales. Esto permite que personas no expertas en programación, obvien de los detalles de implementación de los juegos, y centrarse en aportar el conocimiento relativo a su campo.

Otra de las ventajas de la lógica difusa reside en que permite cambios graduales en los sistemas de control. Por ejemplo, en los juegos de conducción existen las acciones de acelerar, frenar, girar a la derecha y girar a la izquierda. Un controlador no difuso maneja estos conceptos de forma atómica, sin cambios graduales entre ellos. Un NPC conductor con un controlador clásico realizaría giros bruscos o escalonados, lo que reduce la experiencia de juego al no tener sensación de inteligencia por parte de los conductores manejados por el ordenador. La lógica difusa demuestra tener unas características muy adecuadas para esta tarea [TANA92]. El uso de reglas para determinar la velocidad que debe llevar un coche en función de la distancia con el obstáculo más cercano permite experiencias de juego más realistas, al manejar todo el espectro de velocidades posibles (Figura 13).

```
Rule 1: IF Distance is Near THEN Speed Slow

Rule 2: IF Distance is Medium THEN Speed Medium

Rule 3: IF Distance is Far THEN Speed Fast
```

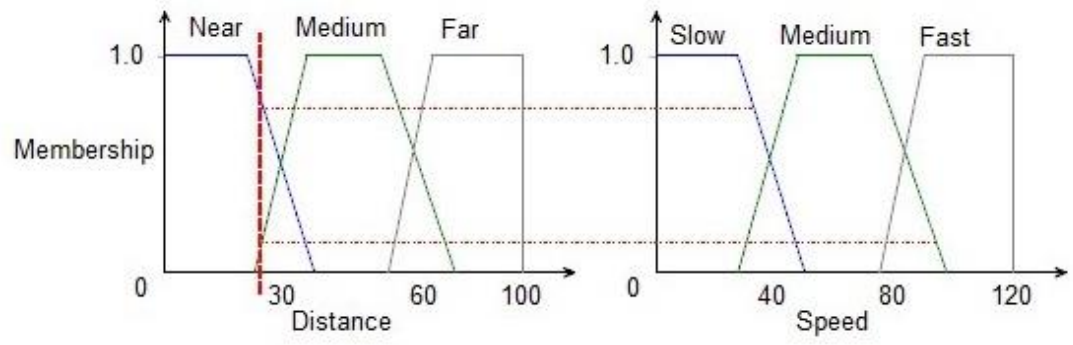


Figura 13 Controlador difuso de la velocidad (en km/h) respecto a distancias (en m) (Figura tomada de http://purvag.com/blog/?p=284)

La lógica difusa también presenta dificultades. Esta técnica requiere la definición de las entradas y las salidas del sistema y las relaciones existentes entre ellas. No tener un conocimiento previo sobre la materia hace que el desarrollo para encontrar el conjunto de reglas y definir cada una de las variables sea un proceso complicado. En el mundo de los juegos, esto supone un incremento en el tiempo de desarrollo y en el coste del proyecto. Además, no se asegura que se obtengan resultados satisfactorios al final. Otro de los inconvenientes de la lógica difusa en los videojuegos está en el tiempo de proceso. Si el juego es muy complejo y el número de reglas es muy elevado, el tiempo para procesar el sistema puede hacer que el juego se vuelva lento y empeore la experiencia de juego.

4.2.1 Máquinas de estados difusas

El principal uso de la lógica difusa en los videojuegos es como parte de las máquinas de estados. La transición entre estados se realiza empleando toma de decisiones en base a reglas difusas [IANM06]. La lógica difusa introduce comportamientos menos predictivos a las máquinas de estados al emplear diversas reglas para disparar un cambio de estado. Esto permite a los desarrolladores tener el potencial estructural que

ofrecen las máquinas de estados y, al mismo tiempo, un sistema menos predecible, y por tanto más entretenido y dinámico que con las máquinas de estados clásicas.

La implementación de las máquinas de estados difusas se puede abordar desde dos puntos de vista distintos, mediante reglas difusas que provoquen transiciones entre estados (disparadores difusos), o haciendo que la pertenencia a un estado sea difusa. Por ejemplo, se desea que un personaje pase de un estado de ataque a uno de defensa en el caso de tener la vida baja. Esto se puede abordar de las dos formas anteriormente mencionadas.

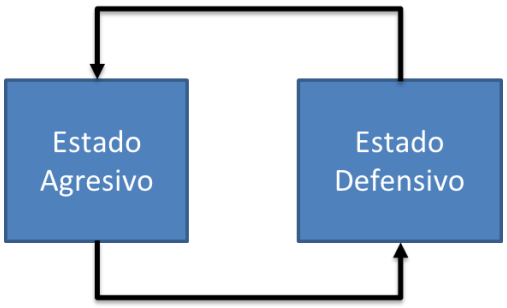


Figura 14 Máquina de estados Agresivo-Defensivo

La implementación mediante disparadores difusos se muestra en la Figura 15. Como se puede observar, cada uno de los estados tiene una base de reglas diferente del resto. Una vez que la variable "Estado_Destino" pertenezca a uno de los estados transicionables, la máquina de estados ejecuta el cambio (Figura 15).

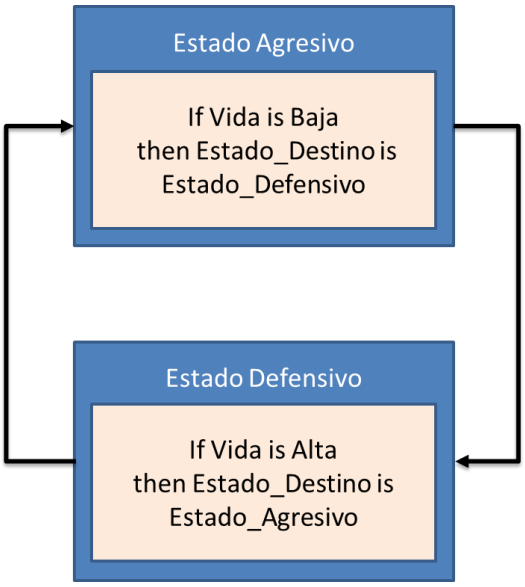


Figura 15 Máquina de estados con disparadores difusos

Por otro lado, las máquinas que emplean estados difusos tienen una única base de reglas que se comprueba constantemente para conocer el estado actual en el que se debe encontrar el sistema en base a sus entradas, en caso del ejemplo, la vida del personaje. En esta ocasión se parte de la máquina de estados de la Figura 14. La transición entre los estados se realiza evaluando el sistema empleando la base de reglas mostrada en la Tabla 3 según la variable "Vida" mostrada en la Figura 16, obteniendo un valor de la variable "Estado" mostrado en la Figura 17.

If Vida is Baja then Estado_Actual is Estado_Defensivo
If Vida is Alta then Estado_Actual is Estado_Agresivo

Tabla 3 Base de reglas de una máquina de estados difusos

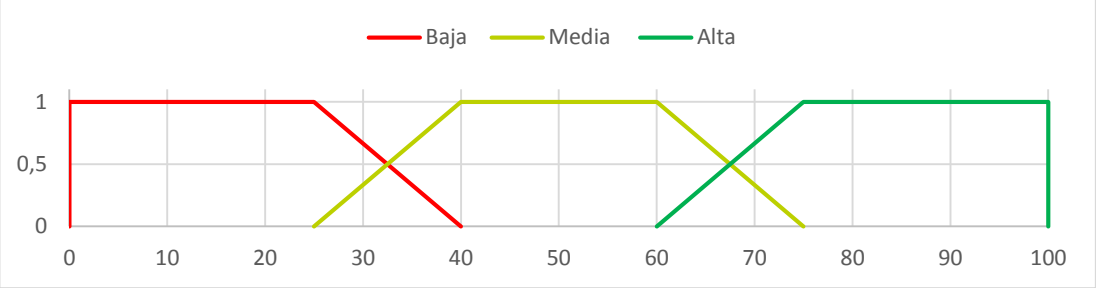


Figura 16 Variable difusa Vida

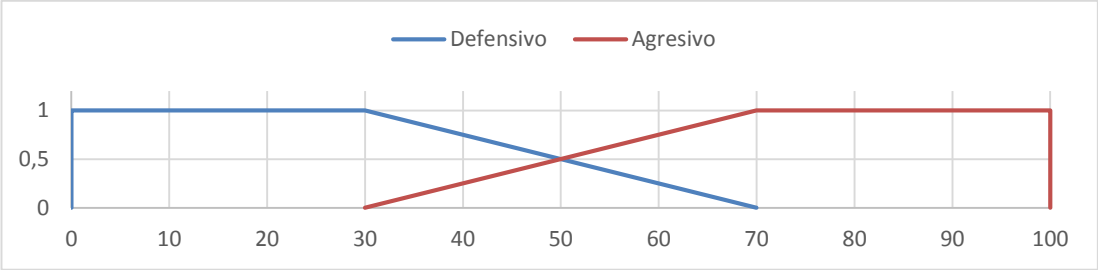


Figura 17 Variable difusa del sistema que representa el estado del personaje, en unidades de agresividad

5 Planteamiento

Los juegos de lucha en 2D han sido siempre muy populares, especialmente en los años 90. Un dato que refleja este hecho es del juego Street Fighter II, el cual es juego arcade con mayores beneficios de todos los tiempos sólo por detrás del Pac-Man y del Space Invaders [USGM13], con unos beneficios en 1995 de más de 2 mil millones de dólares. Gran parte del atractivo se basa en las múltiples opciones que ofrecen este tipo de juego y la destreza requerida a la hora de manejarlas.

Apenas existe en la literatura referencias a videojuegos de lucha donde se emplee lógica difusa [SCHW09]. Donde más se emplea la lógica difusa es como sistema de razonamiento para la transición entre estados en las máquinas de estados finitas, tomando el nombre de máquinas de estados difusas (FSM en inglés). Sin embargo, debido a que no es frecuente encontrar máquinas de estados en juegos de lucha, tampoco lo es encontrar FSMs.

Las principales técnicas de IA aplicadas en juegos de lucha han sido implementaciones adhoc para garantizar una buena experiencia de juego. Actualmente es más común encontrar en la literatura artículos sobre desarrollos de IA en videojuegos. Sin embargo, la mayoría de los juegos 2D más populares son de hace más de 20 años y apenas se realizaban publicaciones al respecto. No obstante, las IAs comúnmente poseían más información y más precisa sobre el estado del juego que el jugador. En el juego Street Fighter II (Capcom, 1991), ciertos movimientos especiales requieren una combinación de botones que incluyen agacharse antes de realizar el movimiento especial. Sin embargo, la IA de este juego era capaz de realizar este movimiento especial sin agacharse primero. Otro ejemplo se puede encontrar en el juego Mortal Kombat (Midway, 1992). En este título existe lo que se conoce como *IA perfecta* [TVTR09]. La IA perfecta tiene las siguientes características: bloquea todos los movimientos bloqueables y esquiva los que no lo sean; nunca ejecuta movimientos como saltar o rodar que puedan suponer un impedimento a un bloqueo necesario; y conoce y ejecuta todos los contra-ataques posibles. Pese a todo, es posible (aunque altamente improbable) ganar. Este tipo de IAs suponen un gran reto a los jugadores muy experimentados, pero, en general, son frustrantes para el resto de jugadores. Como cada personaje tiene unas habilidades especiales en cada juego, los desarrolladores crean IAs únicas para cada uno de los personajes. Esto supone un alto coste en cuanto a recursos. Además, el resultado de este tipo de desarrollo es un juego monótono y previsible. Para evitar, esto es habitual añadir cierto grado de aleatoriedad en las acciones tomadas, pero esto lleva a una dificultad añadida a la hora de depurar el juego al ser muy difícil de controlar los motivos por los cuales un personaje decide realizar cierto movimiento.

Los sistemas basados en reglas difusas tienen una serie de ventajas en cuanto a la facilidad de desarrollo y a la predictibilidad del juego. Gracias al motor de inferencia, la programación de una IA mediante un sistema basado en reglas se realiza mediante una serie de reglas lógicas en lenguaje natural. La proximidad de este lenguaje a la comprensión humana permite a los desarrolladores de este tipo de juegos una mayor facilidad a la hora de depurar las acciones que los personajes toman. Además, es posible generar automáticamente una serie de reglas iniciales que el programador es capaz de entender rápidamente y adaptarlas a las necesidades del producto. Los sistemas basados en reglas simulan un comportamiento más cercano al humano utilizando las mismas entradas al sistema que una persona. De esta forma, el jugador humano no se siente frustrado al ver que el oponente es capaz de realizar acciones que al jugador humano no le están permitidas. Siguiendo por esta línea, la lógica difusa permite al desarrollador simular mucho mejor la imprecisión con la que lidia un jugador humano. Pese a que los valores del juego son perfectamente nítidos, la forma de representación de datos visuales como distancias o la vida restante hace que para el humano ese dato sea totalmente impreciso. Las etiquetas lingüísticas como "cerca" o "lejos" permiten a los desarrolladores crear juegos atractivos sin la necesidad de ir "a nivel de píxel" con las entradas al sistema.

Una de las pocas referencias que emplean la lógica difusa en un videojuego de lucha es de un desarrollo hecho para el mismo *framework* elegido para las pruebas del trabajo de este documento. En esta referencia se emplean dos bases de conocimiento en cascada. La primera sirve para determinar si el predictor Knn que utilizan tiene suficiente información o no. Si se considera que no, se utiliza una pequeña base de conocimiento para determinar si se realiza un ataque de largo o corto alcance. Como se puede observar, el uso que se le da a la lógica difusa y a los sistemas de conocimiento basados en reglas difusas es mucho menos ambicioso del que se propone en este trabajo.

6 Solución

Se propone un sistema inteligente basado en reglas difusas que sea capaz de jugar a videojuegos de lucha donde la base de conocimiento se obtiene mediante un nuevo método basado en programación genética.

El motor de inferencia viene definido e implementado en la librería "fuzzylite" [FUZZ15]. El operador de conjunción elegido es el producto del álgebra de Bool. El operador de disyunción elegido es la suma. Estos operadores son los más empleados y son adecuados para el sistema planteado. La creación de la base de conocimiento es siempre el proceso más complicado a la hora de crear un sistema basado en reglas. En este caso, para obtener el mejor conjunto de reglas, se emplea la programación genética guiada por gramáticas, utilizando dos operadores de cruce y mutación diferentes para comparar sus resultados.

La definición de los conjuntos difusos se realiza en función de las características propias del juego en cuestión. En este caso, al ser un juego de lucha 2D, las variables que maneja el sistema son la distancia entre los oponentes, sus posiciones, su energía (elemento restrictivo para realizar ciertos movimientos) y la puntuación (donde se contempla la vida que cada personaje ha perdido). La acción a tomar es el movimiento que realiza el personaje.

En este proyecto se emplea un conjunto de herramientas, librerías y programas (en adelante *framework*) hechos en Java y orientada a la competición de sistemas inteligentes en un juego de lucha 2D. Este *framework* implementa todo lo necesario para la ejecución del juego y aporta una interfaz en Java que los sistemas inteligentes deben implementar para poder interactuar con el juego.

6.1 Definición de la base de conocimiento y reglas difusas

Un juego de lucha consta de un escenario fijo donde los luchadores pueden moverse libremente. El objetivo del juego consiste en utilizar los movimientos permitidos de los luchadores para restar la mayor cantidad de puntos de vida al oponente y mantener la mayor cantidad de puntos de vida propios.

Los diferentes movimientos permitidos para un jugador vienen establecidos por el creador del juego, así como la distancia desde la que un jugador puede golpear a otro con un movimiento en concreto. Sin embargo, realizar un movimiento de golpeo al

contrario a una distancia adecuada no garantiza que se produzca un impacto debido a que el oponente puede desplazarse por el escenario. Para la resolución de esta cuestión se emplea la lógica difusa. Se conocen con certeza las distancias a las que un movimiento no es efectivo, y a las que al oponente no le da tiempo a desplazarse para evitar el golpe. Sin embargo, existe una región intermedia donde el movimiento puede resultar efectivo o no. Los conjuntos difusos permiten definir posibilidades de efectividad del golpe. De esta forma es posible definir reglas donde la realización de dichos movimientos tiene en cuenta estos factores.

Para definir las posibles acciones de un jugador se definen dos tipos de reglas:

- Reglas de movimiento: Estas reglas definen el tipo de desplazamiento que realiza el jugador. Los movimientos pueden ejecutarse en horizontal, en vertical y de manera conjunta. El eje X representa la idoneidad de ejecutar el movimiento representado con un valor arbitrario de 0 a 1, siendo 0.5 el valor por omisión en el caso de que ninguna regla haga referencia a estas variables. Los tipos de movimiento y sus posibles valores se muestran en la Tabla 4:

Horizontal	**Vertical**
Back	Down
Stand	Stand
Forward	Up

Tabla 4 Etiquetas lingüísitcas de los movimientos

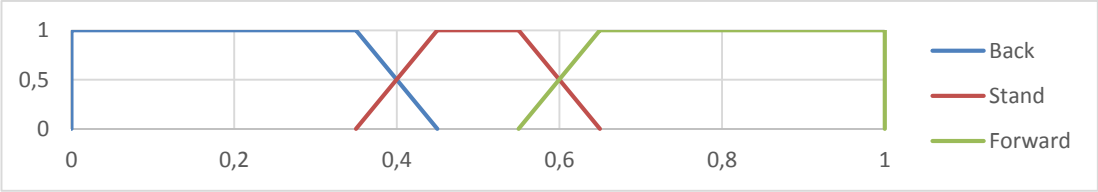


Figura 18 Etiquetas lingüísiticas empleadas en la definición del movimiento horizontal

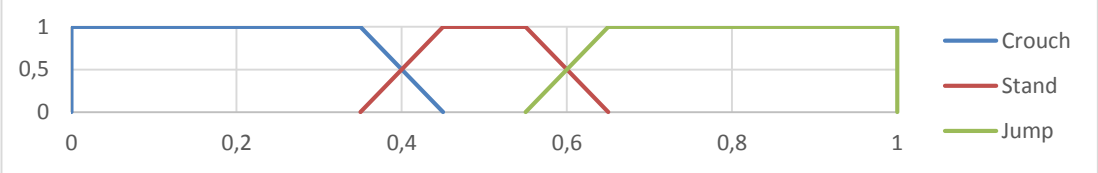


Figura 19 Etiquetas lingüísiticas empleadas en la definición del movimiento vertical

- Reglas de habilidad: Estas reglas definen las combinaciones de botones (combo en adelante) que pueden realizarse. Para ello, el jugador debe cumplir cierta condición de energía acumulada para poder ser realizada. La energía aumenta al golpear o ser golpeado según un valor establecido, dependiendo del movimiento realizado. El uso o no de una habilidad viene definida por 3 conjuntos difusos. Estos conjuntos se muestran en la Figura 20:

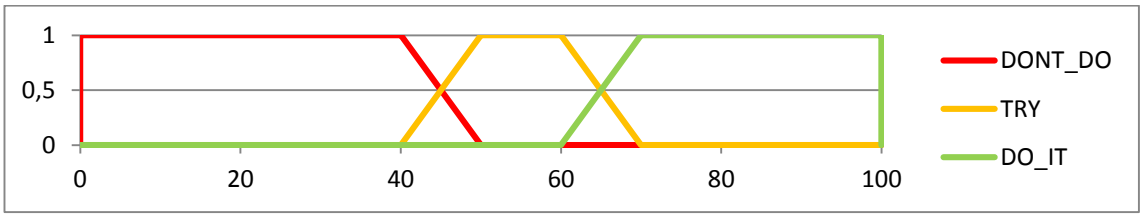


Figura 20 Etiquetas lingüísiticas empleadas en la definición del uso de una habilidad

El eje X representa un valor arbitrario de 0 a 100 que se emplea para determinar la idoneidad de ejecutar cierta habilidad. Cada una está definida por una variable difusa que se evalúa mediante el motor de inferencia. El luchador sólo puede realizar un movimiento o habilidad al mismo tiempo, por lo tanto, aquella con mayor valor a la salida del sistema es la que se ejecuta.

Las reglas en el motor de inferencia utilizado tienen el siguiente formato:

IF *ANTECEDENTES* **THEN** *CONSECUENTES*

donde los antecedentes son el conjunto de variables de entrada al sistema que proporciona el *framework* del juego y los consecuentes son las acciones a realizar por el jugador.

Existen dos tipos de acciones a realizar: las habilidades y los movimientos (desplazamientos). En este proyecto, se ha definido un único consecuente por regla. De esta forma, cada regla interviene exclusivamente a una de las dos posibles acciones. Si la regla es de tipo "movimiento", el consecuente define la dirección del movimiento. En el caso de una regla de tipo "habilidad", el consecuente determina si el combo debe ejecutarse.

El motor de inferencia supone la hipótesis del mundo cerrado, por lo que en el caso de no tener una regla para una habilidad o un movimiento en concreto, establece como valor por defecto "DONT_DO" para las habilidades (Figura 20) y "STAND" para los desplazamientos (Figura 18, Figura 19). La gramática definida no permite reglas cuyos

consecuentes tienen valores “DONT_DO” ni “STAND”, con el objetivo de limitar el espacio de búsqueda a reglas que impliquen una acción.

Los antecedentes de una regla se unen mediante conjunciones de los valores de las siguientes variables:

- Distancia: El conjunto difuso se define mediante la normalización [0,1) relativo a la máxima distancia a la que pueden situarse los dos jugadores. De esta forma cuando los dos luchadores estén lo más cerca posible la distancia es 0 y cuando esté cada uno en un lado opuesto del escenario este valor es 1.

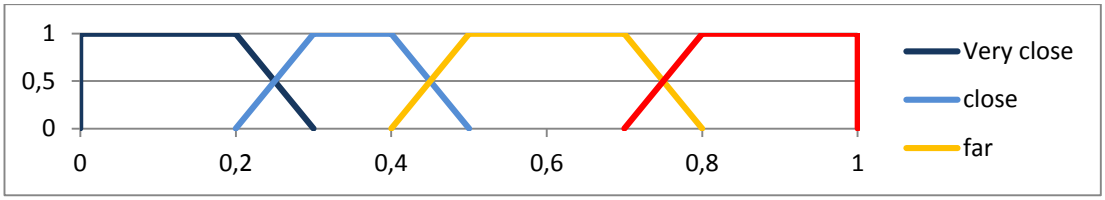


Figura 21 Etiquetas difusas que definen la variable distancia

- Energía propia: Es un valor definido por el juego que sirve de condición para cierto tipo de movimientos especiales y que se obtiene al golpear o ser golpeado durante el combate. Ciertos golpes especiales consumen esta energía al ejecutarse. Es importante destacar que no tiene relación con la vida del personaje. El valor es un entero positivo que representa unidades de energía. Para la variable difusa se establece un valor máximo de 5000 unidades.

- Energía del oponente: Establece la energía que posee el oponente. Al ser condición para los ataques especiales, conocer su valor permite adaptar la estrategia a la posibilidad de la realización de ataques especiales por parte del oponente.

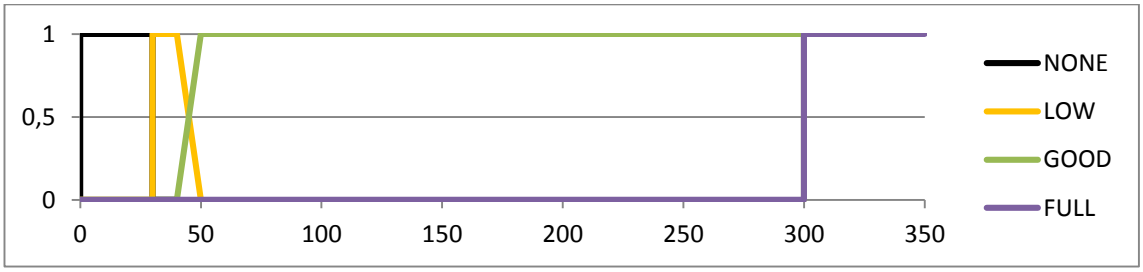


Figura 22 Etiquetas difusas que definen la variable energía. La etiqueta FULL termina en 5000, pero se muestra abierta en la gráfica para una mejor visualización del resto de etiquetas

- Posición propia: Establece la posición propia respecto al centro de la escena. En un juego de lucha es importante dominar el centro de la escena y evitar quedarse acorralado en un lateral. Los valores que se han empleado para la creación de las etiquetas lingüísticas se han obtenido en base al ancho, en píxeles, de la región de juego. El ancho de la zona de juego es de 240 píxeles.

- Posición del oponente: Establece la posición del oponente respecto al centro de la escena. Las etiquetas lingüísticas se establecen como se muestra en la Figura 23.

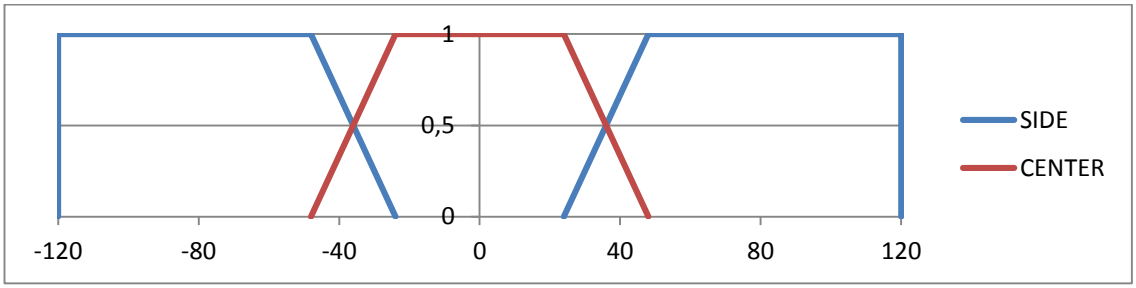


Figura 23 Etiquetas difusas que definen la variable posición

- Acción del oponente: Valores discretos que indican el tipo de movimiento que está realizando el oponente. El conjunto de acciones posibles viene dado por el creador del juego. En el motor de inferencia, estos valores no son difusos, sino un conjunto de valores discretos.

- Puntuación actual: Mide la puntuación del combate en todo momento en base a la vida perdida por ambos luchadores. El sistema de puntuación reparte 1000 puntos entre los dos oponentes, de forma que si uno tiene una puntuación de 400, el otro tiene 600. Los luchadores comienzan con un valor de vida (*HP*, en inglés *Health Points*) 0 y se reduce a valores negativos con cada golpe recibido. La fórmula que determina los valores de la puntuación es la que se muestra en la Ecuación 1 cuando la vida de ambos luchadores es distinta de 0. Cuando ambos tienen vida 0, el valor es 500. Las estrategias más conservadoras cambian el estilo de lucha en función de si van ganando o perdiendo un combate. Las etiquetas lingüísticas empleadas se muestran en la Figura 24.

$$score = \frac{opponentHP}{selfHP + opponentHP}\ x\ 1000$$

Ecuación 1 Sistema de puntuación cuyos valores están en el intervalo [0-1000]

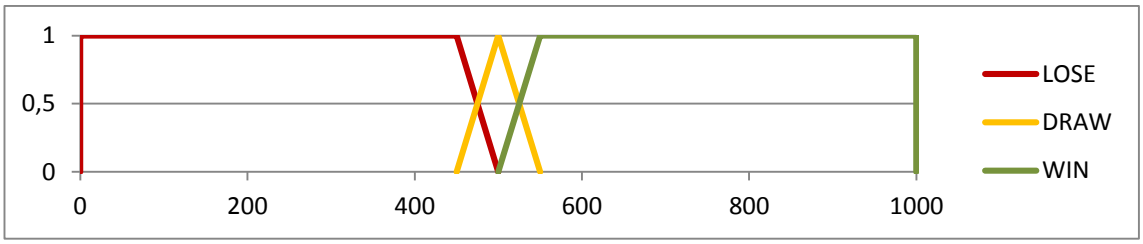


Figura 24 Etiquetas difusas que definen la variable puntuación

6.2 Gramática

Empleando las variables difusas anteriormente descritas, se diseña la siguiente gramática, que define el conjunto de reglas de una base de conocimiento:

```
Axioma: S

No terminales: REGLAS REGLA CONDICIONES ACCION CONDICION
CONDICION_DISTANCIA VALORES_DISTANCIA USO_SKILL SKILL COMBO MOVIMIENTO
MOVIMIENTO_HORIZONTAL MOVIMIENTO_VERTICAL VALORES_HORIZONTAL
VALORES_VERTICAL CONDICION_ENERGIA_OWN CONDICION_ENERGIA_OPP
VALORES_ENERGIA CONDICION_ACCION_OPP VALORES_OPP_ACCION
CONDICION_POSITION_OWN CONDICION_POSITION_OPP VALORES_POSITION
CONDICION_PUNTUACION VALORES_PUNTUACION CONDICIONESIZ CONDICIONESDER
CONDICIONESIZ_I CONDICIONESIZ_D CONDICIONESDER_I

Terminales: ; if then and is DISTANCIA VERY_FAR FAR CLOSE VERY_CLOSE TRY
DO_IT HorizontalMove BACK STAND FORWARD VerticalMove CROUCH STAND JUMP
OWN_ENERGY OPP_ENERGY NONE LOW GOOD FULL NEUTRAL STAND FORWARD_WALK DASH
BACK_STEP CROUCH JUMP FOR_JUMP BACK_JUMP AIR STAND_GUARD CROUCH_GUARD
AIR_GUARD STAND_GUARD_RECOV CROUCH_GUARD_RECOV AIR_GUARD_RECOV
STAND_RECOV CROUCH_RECOV AIR_RECOV CHANGE_DOWN DOWN RISE LANDING THROW_A
THROW_B THROW_HIT THROW_SUFFER STAND_A STAND_B CROUCH_A CROUCH_B AIR_A
AIR_B AIR_DA AIR_DB STAND_FA STAND_FB CROUCH_FA CROUCH_FB AIR_FA AIR_FB
AIR_UA AIR_UB STAND_D_DF_FA STAND_D_DF_FB STAND_F_D_DFA STAND_F_D_DFB
STAND_D_DB_BA STAND_D_DB_BB AIR_D_DF_FA AIR_D_DF_FB AIR_F_D_DFA
AIR_F_D_DFB AIR_D_DB_BA AIR_D_DB_BB STAND_D_DF_FC OPP_ACTION OWN_POSITION
OPP_POSITION CENTER SIDE SCORE LOSE DRAW WIN

Producciones:
S ::= REGLAS
```

```
REGLAS ::= REGLA | REGLAS ; REGLA
REGLA ::= if CONDICIONES then ACCION
CONDICIONES ::= CONDICIONESIZ | CONDICIONESIZ and CONDICIONESDER |
CONDICIONESDER
CONDICIONESIZ ::= CONDICIONESIZ_I | CONDICIONESIZ_I and CONDICIONESIZ_D |
CONDICIONESIZ_D
CONDICIONESIZ_I ::= CONDICION_DISTANCIA | CONDICION_DISTANCIA and
CONDICION_ENERGIA_OWN | CONDICION_ENERGIA_OWN
CONDICIONESIZ_D ::= CONDICION_ENERGIA_OPP |CONDICION_ENERGIA_OPP and
CONDICION_POSITION_OWN | CONDICION_POSITION_OWN
CONDICIONESDER ::= CONDICIONESDER_I | CONDICIONESDER_I and
CONDICION_PUNTUACION | CONDICION_PUNTUACION
CONDICIONESDER_I ::= CONDICION_POSITION_OPP | CONDICION_POSITION_OPP and
CONDICION_ACCION_OPP | CONDICION_ACCION_OPP
CONDICION_DISTANCIA ::= DISTANCIA is VALORES_DISTANCIA
VALORES_DISTANCIA ::= VERY_FAR | FAR | CLOSE | VERY_CLOSE
CONDICION_ENERGIA_OWN ::= OWN_ENERGY is VALORES_ENERGIA
CONDICION_ENERGIA_OPP ::= OPP_ENERGY is VALORES_ENERGIA
VALORES_ENERGIA ::= NONE | LOW | GOOD | FULL
CONDICION_ACCION_OPP ::= OPP_ACTION is VALORES_OPP_ACCION
VALORES_OPP_ACCION ::= NEUTRAL | STAND | FORWARD_WALK | DASH | BACK_STEP
| CROUCH | JUMP | FOR_JUMP | BACK_JUMP | AIR | STAND_GUARD | CROUCH_GUARD
| AIR_GUARD | STAND_GUARD_RECOV | CROUCH_GUARD_RECOV | AIR_GUARD_RECOV |
STAND_RECOV | CROUCH_RECOV | AIR_RECOV | CHANGE_DOWN | DOWN | RISE |
LANDING | THROW_A | THROW_B | THROW_HIT | THROW_SUFFER | STAND_A | STAND_B
| CROUCH_A | CROUCH_B | AIR_A | AIR_B | AIR_DA | AIR_DB | STAND_FA |
STAND_FB | CROUCH_FA | CROUCH_FB | AIR_FA | AIR_FB | AIR_UA | AIR_UB |
STAND_D_DF_FA | STAND_D_DF_FB | STAND_F_D_DFA | STAND_F_D_DFB |
STAND_D_DB_BA | STAND_D_DB_BB | AIR_D_DF_FA | AIR_D_DF_FB | AIR_F_D_DFA |
AIR_F_D_DFB | AIR_D_DB_BA | AIR_D_DB_BB | STAND_D_DF_FC
CONDICION_POSITION_OWN ::= OWN_POSITION is VALORES_POSITION
CONDICION_POSITION_OPP ::= OPP_POSITION is VALORES_POSITION
VALORES_POSITION ::= CENTER | SIDE
CONDICION_PUNTUACION ::= SCORE is VALORES_PUNTUACION
VALORES_PUNTUACION ::= LOSE | DRAW | WIN
ACCION ::= COMBO | MOVIMIENTO
COMBO ::= SKILL is USO_SKILL
SKILL ::= STAND_GUARD | CROUCH_GUARD | AIR_GUARD | THROW_A | THROW_B |
STAND_A | STAND_B | CROUCH_A | CROUCH_B | AIR_A | AIR_B | AIR_DA | AIR_DB
| STAND_FA | STAND_FB | CROUCH_FA | CROUCH_FB | AIR_FA | AIR_FB | AIR_UA
| AIR_UB | STAND_D_DF_FA | STAND_D_DF_FB | STAND_F_D_DFA | STAND_F_D_DFB
| STAND_D_DB_BA | STAND_D_DB_BB | AIR_D_DF_FA | AIR_D_DF_FB | AIR_F_D_DFA
| AIR_F_D_DFB | AIR_D_DB_BA | AIR_D_DB_BB | STAND_D_DF_FC
USO_SKILL ::= TRY | DO_IT
MOVIMIENTO ::= MOVIMIENTO_HORIZONTAL | MOVIMIENTO_VERTICAL
MOVIMIENTO_HORIZONTAL ::= HorizontalMove is VALORES_HORIZONTAL
VALORES_HORIZONTAL ::= BACK | STAND | FORWARD
MOVIMIENTO_VERTICAL ::= VerticalMove is VALORES_VERTICAL
VALORES_VERTICAL ::= CROUCH | STAND | JUMP
```

Tabla 5 Gramática diseñada para el framework del juego

La gramática define un conjunto de producciones separadas por el carácter ";". Cada nueva regla añade un grado más de profundidad al árbol de derivación total.

En la Figura 25 se muestra parte de un posible árbol de derivación basado en la gramática anterior:

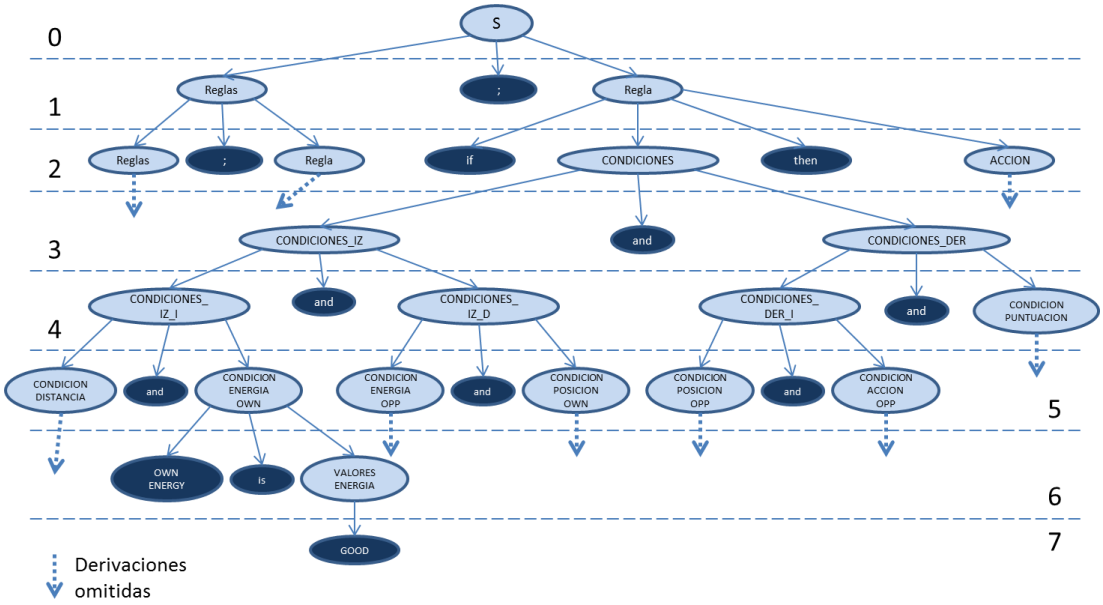


Figura 25 Árbol de derivación donde se observa la profundidad mínima de las derivaciones

Se puede observar que la profundidad mínima de un árbol de derivación corresponde a una base de conocimiento compuesta por una única regla. Este caso tiene una profundidad de 7. Existe un tipo de derivación especial cuya profundidad puede ser 6. Es el de una única regla con sólo la condición de puntuación. La Figura 26 muestra este caso.

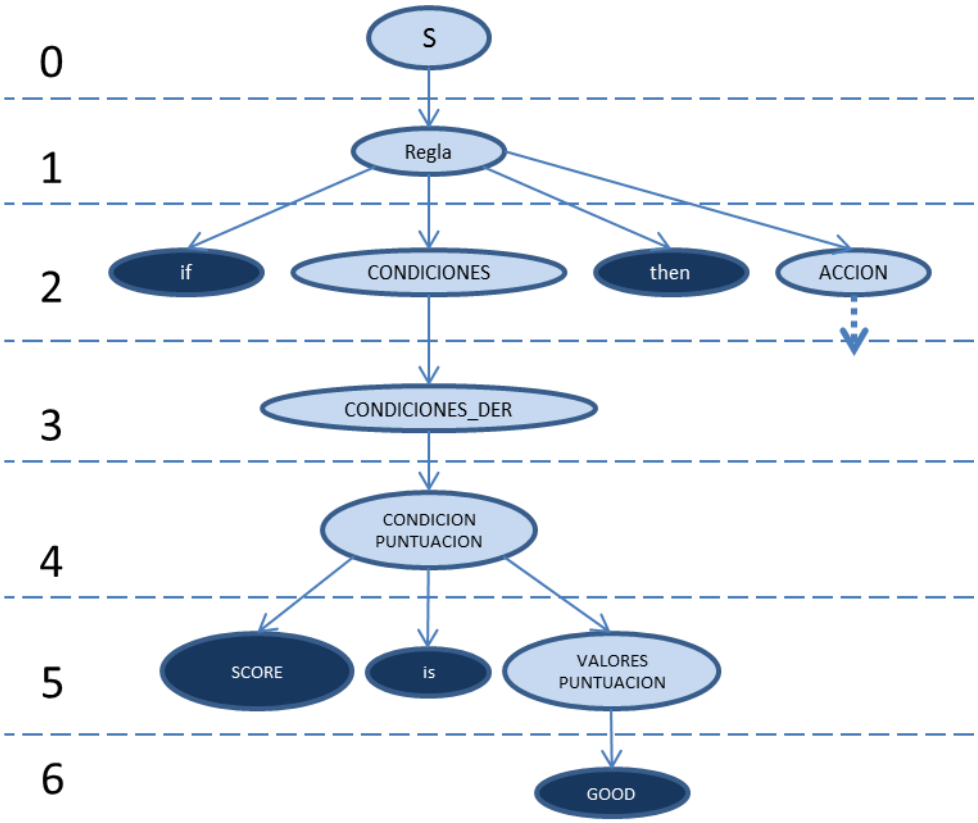


Figura 26 Mínimo árbol de derivación posible

El diseño de esta gramática ha sido pensado para no favorecer ninguna condición en el cruce de Whigham, que es con el que se compara el nuevo método desarrollado. Este cruce selecciona aleatoriamente un nodo no terminal del primer padre elegido para el cruce y lo sustituye por otro del segundo padre cuya raíz tiene el mismo nodo no terminal. Si el diseño de la gramática no se hace teniendo en cuenta esta característica, se pueden dar casos en los que alguna de las variables de entrada sea más probable que se cambie que otras. El siguiente ejemplo muestra esta situación:

```
Axioma: S

Producciones:
S ::= REGLAS

REGLAS ::= REGLA | REGLAS ; REGLA

REGLA ::= if CONDICIONES then ACCION
```

```
CONDICIONES ::= CONDICION_DISTANCIA | CONDICION_DISTANCIA and
CONDICIONES_1 | CONDICIONES_1

CONDICIONES_1 ::= CONDICION_ENERGIA_OWN | CONDICION_ENERGIA_OWN and
CONDICIONES_2 | CONDICIONES_2

CONDICIONES_2 ::= CONDICION_ ENERGIA_OPP | CONDICION_ ENERGIA_OPP and
CONDICIONES_3 | CONDICIONES_3

CONDICIONES_3 ::= CONDICION_POSITION_OWN | CONDICION_POSITION_OWN and
CONDICIONES_4 | CONDICIONES_4

CONDICIONES_4 ::= CONDICION_POSITION_OPP | CONDICION_POSITION_OPP and
CONDICIONES_5 | CONDICIONES_5

.
.
.
```

Tabla 6 Gramática con las derivaciones de las condiciones en cascada

Un árbol de derivación de esta gramática sería como se muestra en la Figura 27:

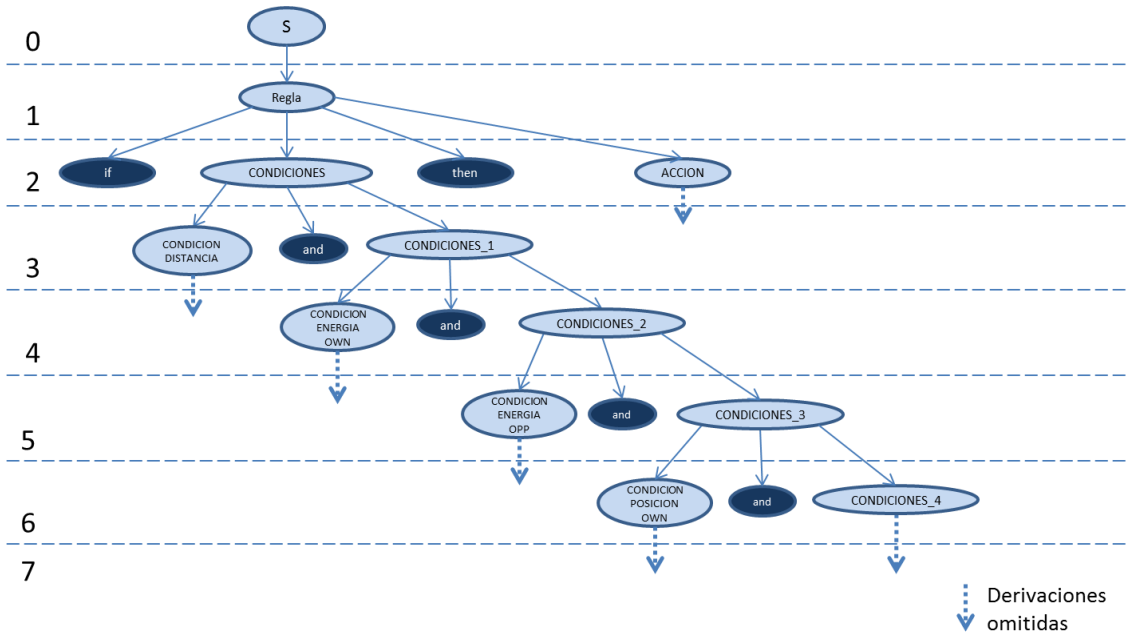


Figura 27 Árbol de derivación de una gramática con las condiciones en cascada

Se propone como ejemplo una elección para un cruce de dos individuos con una única regla:

	Individuos
1	IF DISTANCIA IS FAR AND ENERGIA_OWN IS FULL AND ENERGIA_OPP IS LOW AND POSICION_OWN IS CENTER THEN STAND_A IS DO_IT
2	IF DISTANCIA IS CLOSE AND ENERGIA_OWN IS LOW AND ENERGIA_OPP IS FULL AND POSICION_OWN IS SIDE THEN AIR_B IS DO_IT

Tabla 7 Individuos de ejemplo seleccionados para un cruce Whigham

Los dos individuos del ejemplo tienen árboles de derivación similares como se muestra en la Figura 28:

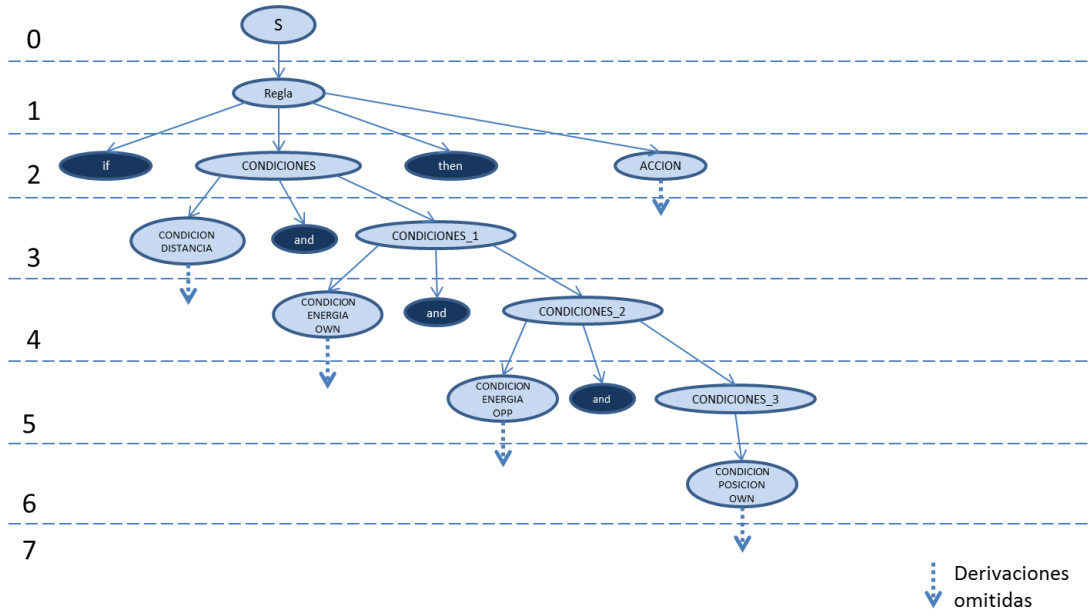


Figura 28Parte común del árbol de derivación de los individuos del ejemplo

En el cruce Whigham se selecciona aleatoriamente un nodo no terminal que se utiliza para intercambiar los subárboles seleccionados entre los individuos. Para que el antecedente de "CONDICIÓN_POSICION" esté en los descendientes con el consecuente del ancestro contrario (por ejemplo, antecedente del individuo 1 con consecuente del individuo 2), el nodo elegido debe ser uno de los que se marcan en naranja la Figura 29:

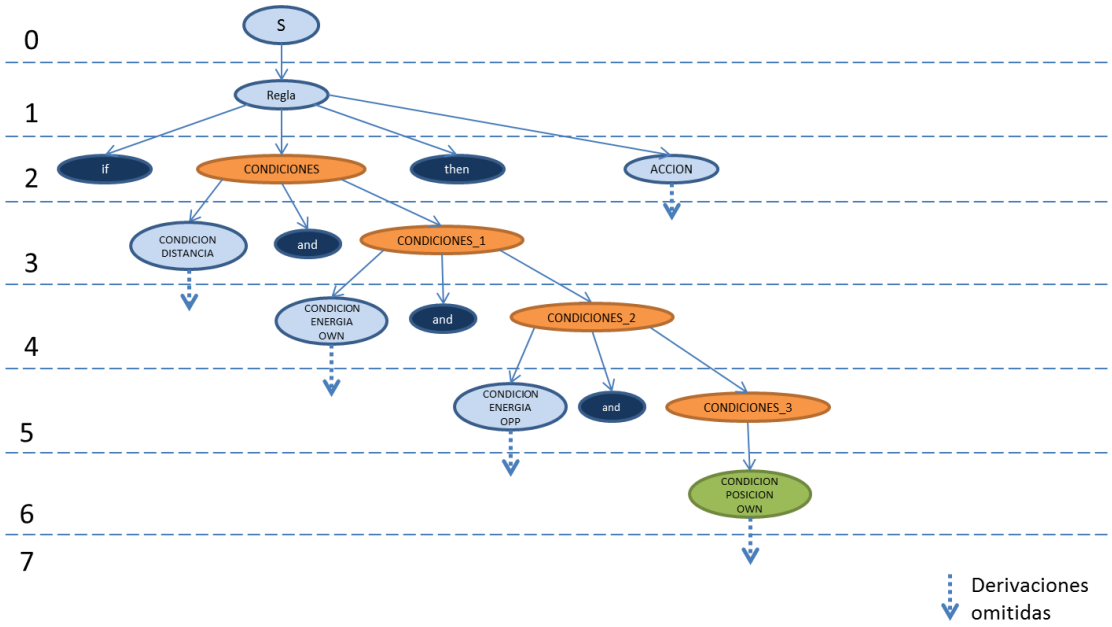


Figura 29 Nodos no terminales que provocan que la condición de posición no afecte al consecuente de la regla a la que pertenece en los ancestros

Para que la "CONDICIÓN_DISTANCIA" cambie el consecuente en los descendientes, es necesario que se seleccione el nodo marcado en naranja (Figura 30):

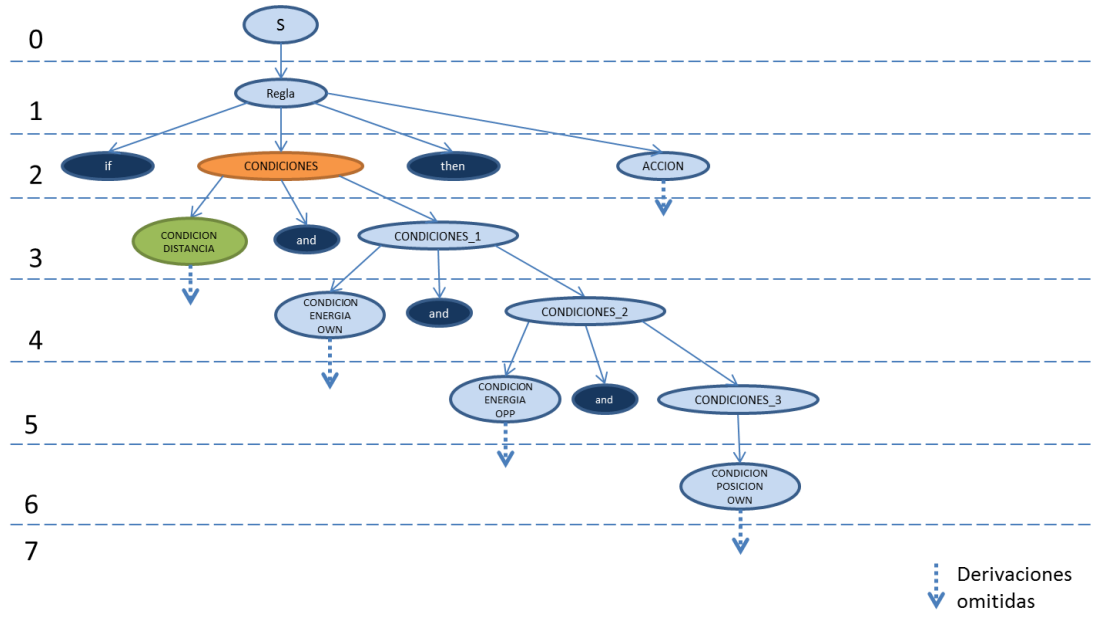


Figura 30 Nodos no terminales que provocan que la condición de distancia no afecte al consecuente de la regla a la que pertenece en los ancestros

En el primer individuo del ejemplo de la

Tabla 7, el antecedente que corresponde a la condición de posición propia es "POSICION_OWN IS CENTER" y su consecuente es "STAND_A IS DO_IT". Al intercambiar mediante el cruce Whigham cualquier nodo no terminal marcado en naranja en la Figura 29, en los hijos, la condición "POSICION_OWN IS CENTER" está siempre con el consecuente "AIR_B IS DO_IT" del individuo contrario. La condición "DISTANCIA IS FAR" del primer individuo estará en los descendientes con el consecuente "AIR_B IS DO_IT" únicamente si el no terminal seleccionado para el cruce es "CONDICIONES", como se muestra en la Figura 30. Se toma como ejemplo la elección del nodo no terminal "CONDICIONES_1" para el cruce de los individuos de ejemplo. El resultado del cruce queda como se muestra en la Tabla 8:

	Ancestros	Descendientes
1	IF DISTANCIA IS FAR AND ENERGIA_OWN IS FULL AND ENERGIA_OPP IS LOW AND POSICION_OWN IS CENTER THEN STAND_A IS DO_IT	IF DISTANCIA IS FAR AND ENERGIA_OWN IS LOW AND ENERGIA_OPP IS FULL AND POSICION_OWN IS SIDE THEN AIR_B IS DO_IT
2	IF DISTANCIA IS CLOSE AND ENERGIA_OWN IS LOW AND ENERGIA_OPP IS FULL AND POSICION_OWN IS SIDE THEN AIR_B IS DO_IT	IF DISTANCIA IS CLOSE AND ENERGIA_OWN IS FULL AND ENERGIA_OPP IS LOW AND POSICION_OWN IS CENTER THEN STAND_A IS DO_IT

Tabla 8 Resultado del cruce Whigham al seleccionar el no terminal "CONDICIONES_1" en los individuos de ejemplo

La gramática diseñada en este trabajo, y mostrada en Anexo 2, trata de evitar esta dificultad. En la Figura 31, el no terminal "CONDICIÓN_DISTANCIA" termina con un consecuente distinto si se selecciona uno de los no terminales "CONDICIONES", "CONDICIONES_IZ" O "CONDICIONES_IZ_I". El no terminal "CONDICION_POSICION_OWN" puede terminar con otro consecuente si se

selecciona uno de los no terminales “CONDICIONES”, “CONDICIONES_IZ” O “CONDICIONES_IZ_D”. Cada condición puede cambiar de consecuente si se selecciona uno de 3 no terminales de los 5 que afectan a las condiciones. El único no terminal que únicamente depende de 2 no terminales es “CONDICIÓN_PUNTUACIÓN”. La Figura 31 muestra lo anteriormente expuesto.

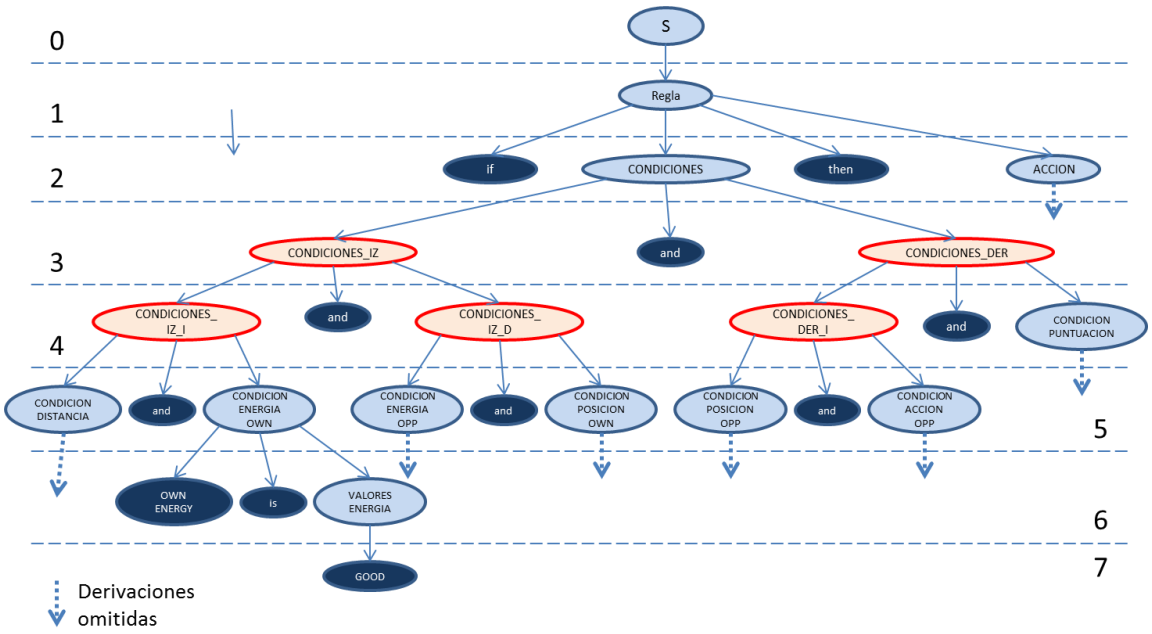


Figura 31 Árbol de derivación de la gramática diseñada en este trabajo donde se resaltan en rojo los nodos no terminales que afectan parcialmente a las condiciones de una regla

Como se puede apreciar, el diseño de la gramática afecta mucho al cruce Whigham. Si ciertas condiciones tienen más probabilidad de cambiar que otras, el rendimiento del PGGG puede verse afectado de manera significativa, debido a que no todas las soluciones del problema son igualmente probables de obtener. Los operadores de mutación y cruce están, por tanto, sesgados a la hora de favorecer la aparición de una condición en una regla o modificar una ya existente.

6.3 Método de la sedimentación

El método de la sedimentación se refiere a nuevos operadores de cruce y mutación definidos en esta TFM que reciben su nombre del proceso de sedimentación que se produce en los lechos marinos, donde las sustancias disueltas en el agua se van asentando en el fondo hasta formar un sustrato sólido. De forma análoga, en una

población de individuos, los mejores conjuntos de reglas que garanticen el buen rendimiento de la base de conocimiento son los que deben ir formando ese estrato sólido de reglas válidas. El resto de reglas no consolidadas aún, se mueven por los individuos como lo hacen los elementos disueltos en el agua. Con el objetivo de evitar que se asienten reglas que no aportan valor a la base de conocimiento, y para evitar una falta de diversidad genética con el tiempo, se proponen dos métodos de mutación: adición y reducción. De esta forma, el nuevo método de la sedimentación propone 3 nuevos operadores para obtener bases de conocimiento; el cruce, la adición y la reducción.

6.3.1 Operador de cruce

El operador de cruce de la sedimentación emplea dos padres para generar dos hijos. En este proceso se seleccionan las reglas que tienen ambos progenitores en común y se añaden al conjunto aún vacío de los descendientes. Posteriormente, se reparten aleatoriamente las reglas diferentes de los progenitores entre los hijos. La Figura 32 muestra el proceso en su conjunto.

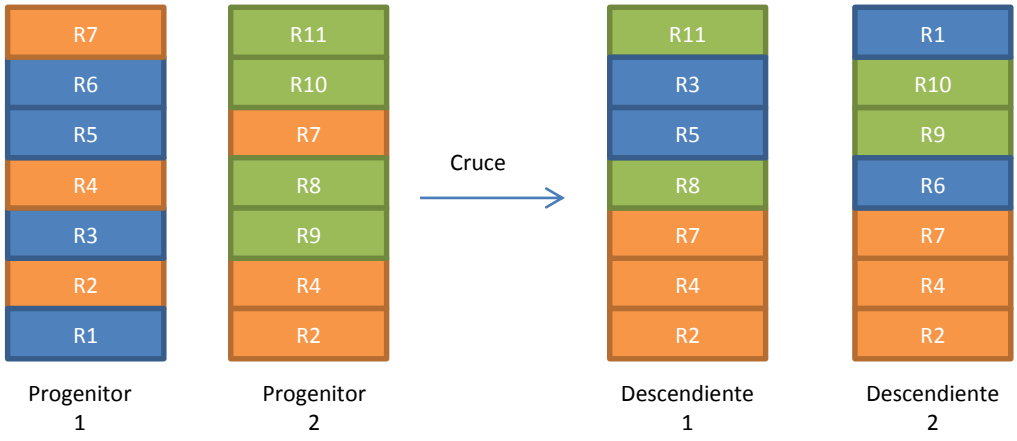


Figura 32 Cruce mediante sedimentación

Como se puede observar, las reglas que son comunes a los dos progenitores, en el ejemplo R2, R4 y R7, van quedando en el fondo de la base de conocimiento de la descendencia. Este cruce garantiza que no existen individuos con reglas duplicadas. También permite obtener los conjuntos de reglas que mejor funcionan en conjunto: si las reglas R1 y R3 del ejemplo hacen que el individuo funcione mejor cuando están

juntas, ninguno de los dos descendientes es mejor que el primer progenitor, debido a que cada descendiente tiene sólo una de ellas.

Este método de cruce también garantiza el "cierre". Mediante este proceso nunca aparecen nuevas reglas, ni la base de conocimiento reduce su tamaño. En el caso de que un progenitor tenga más reglas que el otro, el resultado son dos descendientes con un número de reglas igual a la media de sus progenitores. En caso de que la media no sea un número entero, uno de los descendientes tiene una regla más.

Este cruce no depende de la forma de la gramática empleada para la generación de las reglas al no trabajar sobre los nodos sino sobre reglas completas.

Este cruce no garantiza en ningún caso la diversidad genética ni alcanzar un óptimo que no esté contemplado dentro de la combinación del conjunto de reglas existentes en toda la población, debido a que las reglas únicamente cambian de individuo en cada iteración. Por este motivo, el método de la sedimentación requiere de otros operadores como la adición, la reducción y la selección elitista. Este último permite que, para una regla en concreto, ésta se duplique en los 2 individuos en lugar de aparezca sólo en 1, por lo que aumenta su presencia en el conjunto de la población.

6.3.2 Operador de adición

Este operador añade una regla generada de forma aleatoria a la base de reglas de un individuo. La adición añade diversidad genética en la población y permite explorar bases de conocimiento de mayor tamaño.

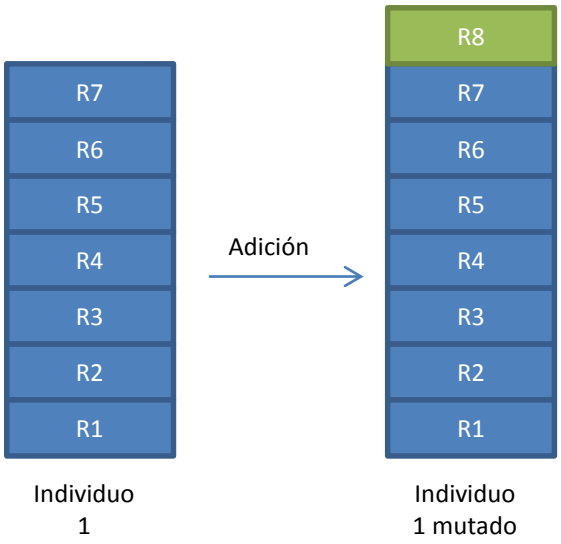


Figura 33 Operador de adición en la sedimentación

6.3.3 Operador de reducción

El operador de reducción de la sedimentación consiste en eliminar una regla de forma aleatoria del individuo seleccionado. Este operador tiene como objetivo eliminar reglas residuales a largo plazo. Estas reglas son las que aportan poco o ningún valor añadido a la base de conocimiento pero que han estado presentes en los mejores individuos de generaciones anteriores por casualidad. Por ejemplo, si un individuo muy bien adaptado tiene una regla que no se dispara nunca en el motor de inferencia, dicha regla se extiende por la población pese a no aportar nada. Este operador permite explorar soluciones con menos reglas, es decir, bases de conocimiento con menos reglas y un alto grado de adaptación. Un motor de inferencia es más rápido cuantas menos reglas tenga que procesar. Este proceso permite llegar precisamente a soluciones con menos reglas, lo que a largo plazo resulta en la eliminación de aquellas que no aportan conocimiento útil a los individuos.

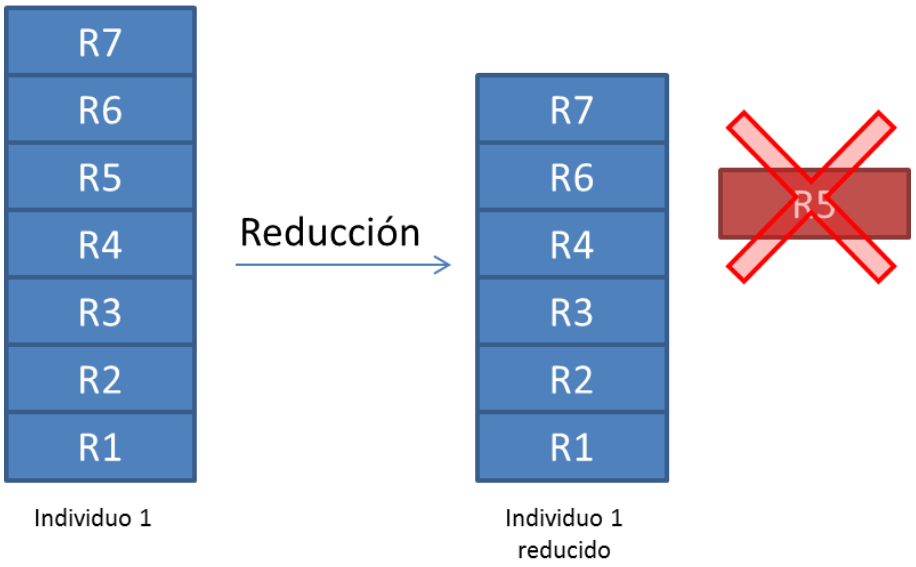


Figura 34 Operador de reducción en la sedimentación

6.3.4 Operador de selección

El operador de selección juega un papel importante en el método de la sedimentación. Para garantizar que las reglas más útiles se encuentran cada vez en más individuos, es necesario que los operadores de adición y reducción mantengan tanto el individuo original como el mutado. Si los individuos mutados están bien adaptados, tienen mayor probabilidad de reproducirse y por tanto las reglas que mayor conocimiento aportan se extienden por la población.

Se recomienda que los individuos sometidos a los operadores de reducción y adición sean elegidos de forma elitista para explorar soluciones con más o menos reglas a partir de aquellas bases de conocimiento que mejor funcionan.

6.4 Implementación

Se ha implementado un algoritmo evolutivo basado en el método de la sedimentación propuesto para determinar la mejor base de conocimiento que obtenga el mejor luchador posible, donde se emplea una gramática únicamente para la generación aleatoria de reglas. También se ha hecho lo mismo empleando PGGG con el cruce Whigham para comparar empíricamente los resultados.

El algoritmo evolutivo implementado mantiene los 4 mejores individuos de la población para la siguiente generación. Estos 4 individuos son los elegidos para los operadores de adición y reducción. De esta forma, se garantiza el correcto funcionamiento del proceso de sedimentación. Se incluyen 2 individuos nuevos generados aleatoriamente como proceso de inmigración. De esta manera, se añade cierta diversidad genética para la siguiente iteración. El resto de individuos se obtienen mediante cruce. El método de selección elegido para el cruce es el de la ruleta. Este método permite que pueda salir cualquier individuo y aporte diversidad en las bases de conocimiento, pero siempre manteniendo preferencia sobre los mejores individuos.

6.4.1 Características del *framework* de lucha empleado

El juego donde se han aplicado estas nuevas técnicas evolutivas es un videojuego de lucha clásico en 2D donde dos luchadores ejecutan movimientos para reducir la mayor cantidad de puntos de vida posible al oponente. La fórmula que determina los valores de la puntuación es la que se muestra en la Ecuación 2 cuando la vida de ambos luchadores es distinta de 0. En ese caso, el valor es 500. Cada combate consta de 3 rondas de 60 segundos, donde la puntuación de cada luchador viene definida por el creador del juego según la función:

$$score = \frac{opponentHP}{selfHP + \ opponentHP} \ x \ 1000$$

Ecuación 2 Cálculo de la puntuación

Como consecuencia de la función anterior, al terminar un combate, cada luchador tiene 3 puntuaciones entre 0 y 1000.

Este juego tiene la particularidad de que los jugadores no se quedan sin vida. Ambos comienzan con 0 y cada golpe del contrario resta cierta cantidad de puntos. Por tanto, en el transcurso del combate, los luchadores tienen valores de vida negativos. El jugador que menos vida haya perdido gana. Esto provoca que las estrategias tiendan a ser más conservadoras.

6.4.2 Algoritmo evolutivo basado en el método de la sedimentación

El algoritmo evolutivo implementado tiene las siguientes características:

- Cuenta con una población inicial de 30 individuos generados aleatoriamente, que codifican bases de conocimiento de reglas difusas.
- Cada individuo de la población inicial tiene 5 reglas.
- El número máximo de iteraciones del algoritmo es de 300.

Cada individuo de la población se enfrenta a 4 oponentes. Los dos primeros son sistemas inteligentes que participaron en la competición "*Fighting Game AI Competition*" [ICEC15] de 2014, siendo uno de ellos el ganador de dicho año. De esta forma, el sistema propuesto busca bases de conocimiento que permiten derrotar a los sistemas inteligentes de ese año. El tercer luchador es una ejecución aleatoria de acciones de un personaje. El objetivo es garantizar la resistencia a la incertidumbre en los movimientos del oponente. Si no se añadiese, el sistema propuesto buscaría bases de conocimiento que ganen a los sistemas anteriores y no alcanzaría resultados que pudieran servir para luchadores desconocidos. El último oponente está constantemente parado sin hacer nada. Con este luchador se premia a aquellos individuos que tienen reglas ofensivas y que se acerquen al rival, debido a que al principio del combate los luchadores están separados. Este oponente se añade con el objetivo de evitar la aparición de individuos que únicamente se dediquen a golpear, esperando a que el oponente se acerque. De esta forma se obtienen luchadores más divertidos.

Para calcular la adaptación (*fitness*) de los individuos, cada uno de ellos se enfrenta en 3 rondas contra cada oponente. De esta forma se obtienen 12 combates para el cálculo del *fitness*. Dicha función es la siguiente:

$$Fitness = MediaCombates - \frac{DesviaciónTípicaCombates}{2} - numReglas * \rho$$

La media de los combates establece lo buen luchador que es el individuo. Este es el principal factor de la medida de adaptación. Su rango se encuentra en [0,1000].

La desviación típica de los combates es un factor que reduce el *fitness*. De esta forma, se busca que los individuos tengan un comportamiento estable. Debido al carácter aleatorio de los combates, la suerte influye en cierto grado en el resultado de cada ronda. Para mitigar este factor aleatorio se penaliza en el *fitness* un alto grado de diversidad en los resultados de los combates.

El último parámetro de la función de adaptación ($numReglas * \rho$) tiene en cuenta el número de reglas del individuo. Con el objetivo de obtener bases de conocimiento cuya carga computacional no sea excesiva, se penaliza un número elevado de reglas. La penalización se establece multiplicando el número de reglas por ρ, siendo éste un factor de penalización por regla. El valor de ρ se ha establecido empíricamente en 10, que se corresponde con un 1% del valor máximo posible del nivel de adaptación.

El parámetro ρ también se puede modificar con el fin de favorecer un número determinado de reglas. Asignando un número negativo al factor ρ, se pueden conseguir individuos que sean favorecidos por tener un mayor número de reglas en su base de conocimiento. Con ello, se puede conseguir que un individuo tenga un abanico mucho mayor de elecciones para enfrentarse a cualquier oponente. En el caso de desarrollar un NPC para el videojuego, se pueden obtener individuos mucho más divertidos contra los que jugar. Teniendo en cuenta que la entrada al sistema de juego mediante los controles se realiza una vez cada 16 milisegundos, una base de conocimiento muy grande supone que el individuo sea un jugador muy lento, y pierda gran parte de sus combates. Las propias mecánicas del juego y del sistema propuesto permiten obtener individuos con el número de reglas óptimo para esos 16 milisegundos de los que se dispone.

7 Resultados

Los dos algoritmos de PGGG con los que se comprueban los resultados del algoritmo evolutivo de la sedimentación tienen las mismas características que el sistema propuesto (población de 30 individuos y 300 iteraciones en total). La diferencia entre estos dos PGGG está en la gramática. En el primero, la gramática está optimizada para evitar el sesgo del cruce Whigham. En el segundo, la gramática no está optimizada para reducir dicho sesgo. Ambas gramáticas se pueden encontrar en los anexos 1 y 2 de este documento.

7.1 Fitness

La Figura 35 muestra la evolución del fitness del mejor individuo de cada población en los 3 algoritmos durante las 300 iteraciones, ejecutando cada algoritmo 35 veces.

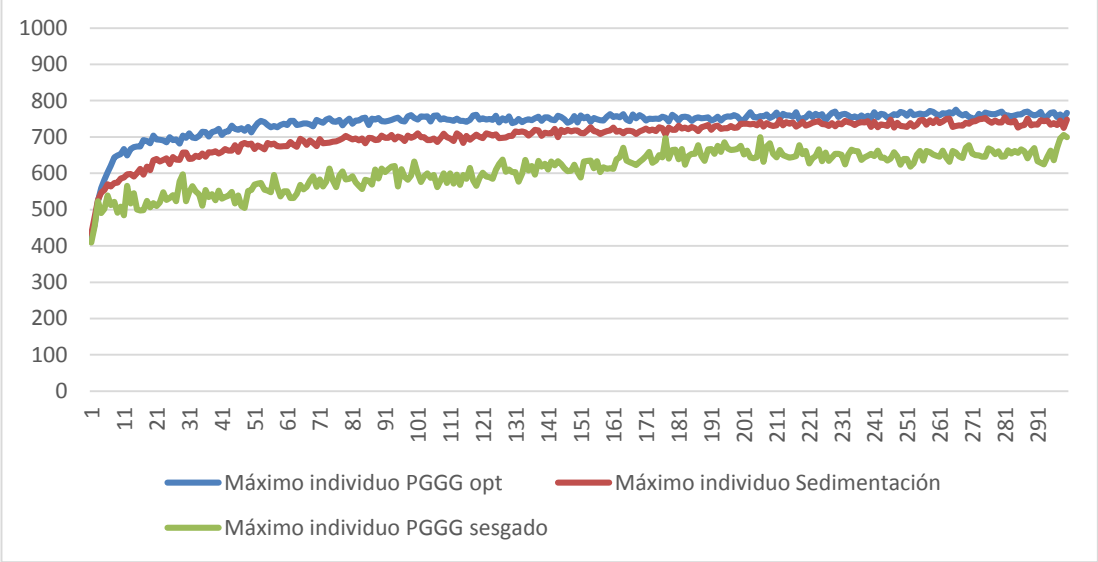


Figura 35 Evolución del mejor individuo de cada población en cada algoritmo del estudio

Se puede observar en las gráficas que no son monótonas crecientes, es decir, el mejor individuo de una generación puede tener peor fitness que el mejor individuo de una generación anterior. Esto muestra la aleatoriedad que existe en los combates que se realizan para medir el nivel de adaptación de los individuos, es decir: un mismo individuo puede obtener diferentes puntuaciones de fitness al ser evaluado en ocasiones distintas. Los resultados muestran una gran diferencia en la convergencia entre el PGGG de la gramática sesgada, el algoritmo de sedimentación y la PGGG con la gramática optimizada. La diferencia entre el primero y los otros dos es de

aproximadamente 300 puntos. Sin embargo, entre los dos últimos, los resultados son muy parejos. La principal diferencia que se observa es que el PGGG optimizado converge sobre la iteración 20, mientras que la sedimentación lo hace sobre las 80 iteraciones.

Para comprobar si son diferentes las soluciones obtenidas por la PGGG óptima y el algoritmo de la sedimentación, se realiza la ANOVA de los resultados obtenidos en 35 ejecuciones completas de ambos algoritmos. En la Tabla 9 se muestran los resultados.

RESUMEN

Grupos	*Cuenta*	*Suma*	*Promedio*	*Varianza*
Fitness mejor individuo PGGG	35	26828	766,514286	3554,55126
Fitness mejor individuo sedimentación	35	26178	747,942857	2794,29076

ANÁLISIS DE VARIANZA

Origen de las variaciones	*Suma de cuadrados*	*Grados de libertad*	*Promedio de los cuadrados*	*F*	*Probabilidad*	*Valor crítico para F*
Entre grupos	6035,7142	1	6035,71429	1,9013591	0,1724452	3,9818962
Dentro de los grupos	215860,62	68	3174,42101			
Total	221896,34	69				

Tabla 9 ANOVA sobre los mejores individuos obtenidos en 30 ejecuciones de los algoritmos de sedimentación y PGGG con la gramática optimizada

La ANOVA muestra que el valor de F está por debajo de la probabilidad crítica, por lo que no se puede rechazar la hipótesis nula, es decir, los dos algoritmos no generan individuos con fitness diferentes.

Se eligen 5 individuos de algoritmo evolutivo para testear las soluciones. Cada individuo realiza 300 combates contra cada uno de los 3 sistemas de control. Estos sistemas son: el ganador del concurso de 2014 [ICEC15], un sistema cuyos movimientos son aleatorios y un sistema del concurso de 2014 [ICEC15] no empleado en el entrenamiento. Los resultados de la Tabla 10 muestran las medias de los combates de los individuos, agrupados según el algoritmo evolutivo con el que se obtuvieron.

Ganador de 2014		Sparring Movimientos Aleatorios		IA no usada en el entrenamiento	
PGGG OPT	*Sedimentación*	*PGGG OPT*	*Sedimentación*	*PGGG OPT*	*Sedimentación*
460,906667	575,2013333	554,204	586,7973333	321,683333	340,1706667

Tabla 10 Media de los resultados de los combates de los 5 individuos elegidos de cada algoritmo evolutivo según el sistema contra el que se prueban las soluciones

Debido a que la puntuación de los combates es mediante un reparto de 1000 puntos entre los luchadores, los resultados obtenidos muestran que los individuos obtenidos mediante sedimentación obtienen mejores resultados en los combates realizados. Los resultados contra el ganador de 2014 muestran que los individuos obtenidos mediante sedimentación consiguen, en media, ganar a este rival, mientras que los obtenidos mediante PGGG, en media, no lo logran. También se observa el descenso en las victorias contra el sistema no utilizado en el entrenamiento. Aumentar el número de sistemas contra los que entrenar permitiría obtener mejores resultados.

Tabla 11 muestra las desviaciones típicas de los resultados de los combates, agrupados por el algoritmo evolutivo con el que se obtuvieron las soluciones.

Ganador de 2014		Sparring Movimientos Aleatorios		IA no usada en el entrenamiento	
PGGG OPT	*Sedimentación*	*PGGG OPT*	*Sedimentación*	*PGGG OPT*	*Sedimentación*
252,523281	171,9247807	227,3171979	223,3512871	213,276327	239,8585589

Tabla 11 Desviaciones típicas de los resultados de los combates de los 5 individuos elegidos de cada algoritmo evolutivo según el sistema contra el que se prueban las soluciones

Los resultados muestran que los resultados de los combates de los individuos obtenidos mediante sedimentación son más estables que los obtenidos con PGGG, especialmente con el sistema utilizado en el entrenamiento. La función de evaluación empleada para obtener los individuos tiene un factor que reduce el nivel de adaptación según la desviación típica de los combates realizados. Los datos incluidos en la Tabla 11 muestran que los individuos obtenidos mediante sedimentación logran mejor este objetivo.

7.2 Reglas

Otro aspecto interesante de los resultados obtenidos son las reglas de las bases de conocimiento. Aplicando una ANOVA al número de reglas del individuo óptimo de los algoritmos de sedimentación y PGGG optimizado se obtiene el resultado mostrado en la

Tabla 12.

RESUMEN

Grupos	*Cuenta*	*Suma*	*Promedio*	*Varianza*
Numero de reglas PGGG	35	97	2,77142857	2,1226890
Número de reglas Sedimentación	35	175	5	11,117647

ANÁLISIS DE VARIANZA

Origen de las variaciones	*Suma de cuadrados*	*Grados de libertad*	*Promedio de los cuadrados*	*F*	*Probabilidad*	*Valor crítico para F*
Entre grupos	86,91428571	1	86,9142857	13,128712	0,00055613	3,981896256
Dentro de los grupos	450,1714286	68	6,62016807			
Total	537,0857143	69				

Tabla 12 ANOVA del número de reglas del individuo óptimo

A tenor de este análisis, se puede confirmar que el número de reglas utilizado en la sedimentación es mayor que en el PGGG. En el caso del PGGG no optimizado, el número medio de reglas en los mejores individuos es de 5,4.

En cuanto al tipo de antecedentes más usados en las reglas obtenidas, la Tabla 13 muestra el porcentaje de aparición de las variables de entrada por regla.

	PGGG Optimizada	PGGG No optimizada	Sedimentación
DISTANCE	25,664%	4,545%	34,783%
OWN_ENERGY	1,770%	13,636%	32,850%
OPP_ENERGY	3,540%	72,727%	35,266%
OPP_ACTION	4,425%	9,091%	28,986%
OWN_POSITION	16,814%	9,091%	37,198%
OPP_POSITION	49,558%	0,000%	44,444%
SCORE	6,195%	27,273%	28,502%
Media de antecedentes por regla	1,079646018	1,363636364	2,420289855
Media Reglas Por Individuo	2,825	5,5	5,175

Tabla 13 Aparición de los antecedentes en las reglas por cada algoritmo

Se puede observar que las características propias de cada algoritmo influyen notablemente en qué variables de entrada son utilizadas por las soluciones encontradas.

La PGGG optimizada obtiene reglas de un único antecedente en la mayoría de los casos. El número medio de reglas por individuo es de casi tres, por lo que, al generar reglas de una única condición, se emplean 3 variables de entrada en la mayoría de los casos. Las tres variables que más aparecen en las reglas de las soluciones encontradas son la posición del oponente, la distancia y, en menor medida, la posición propia. Dada la rápida convergencia del algoritmo expuesta anteriormente, se observa que la PGGG tiende a quedarse en óptimos locales dados por un par de antecedentes.

La PGGG no optimizada tiene un número de reglas alto, sin embargo, los individuos tienen reglas duplicadas en numerosos casos. Además, se observa el efecto del sesgo del cruce Whigham al haber diseñado una gramática en cascada. El orden de derivación de los antecedentes es, de menor a mayor según la profundidad del mínimo árbol de derivación necesario para obtenerlos: "SCORE", "OWN_ENERGY", "OPP_ENERGY", "OPP_ACTION", "OWN_POSITION", "OPP_POSITION" y "DISTANCE". Pese a que las 3 primeras son las menos utilizadas por la PGGG optimizada, son éstas las variables que más utiliza la PGGG con la gramática con diseño en cascada. Se observa que en ninguna solución aparece la variable referente a la posición propia, que es la más utilizada en los otros dos algoritmos. Esto es debido a que, al ser una de las últimas producciones en ser derivada, resulta complicado que un individuo la genere de forma aleatoria. Además, este antecedente casi siempre cambiará de consecuente en un cruce al ser una de las últimas producciones en la gramática en cascada.

En el algoritmo de la sedimentación se utilizan todas las variables de entrada al sistema de forma mucho más uniforme que en los casos anteriores. Sin embargo, sí coincide con la PGGG optimizada en el consecuente más frecuente, la posición del oponente. De la misma forma, los antecedentes referidos a la puntuación y la acción del oponente son los menos utilizados. La sedimentación obtiene reglas con una media antecedentes superior a 2, es decir, en términos de diversión, reglas mucho más complejas, variadas y entretenidas. Como se ha visto anteriormente, este método converge más lentamente, sin embargo, los resultados obtenidos son mucho más variados aunque todos ellos con un nivel de adaptación igualmente alto. Esto demuestra un nivel de exploración mucho más alto que en el caso de la PGGG.

En más de un 20% de los individuos obtenidos como óptimos en la PGGG no optimizada, se obtienen bases de conocimiento con reglas duplicadas. Este valor está entorno al 2% en la PGGG optimizada. El método de la sedimentación no permite este tipo de redundancias, por lo que ningún individuo obtenido mediante este algoritmo tiene reglas duplicadas. En el Anexo 3 de este documento se muestran tres ejemplos de los individuos obtenidos tras las 300 iteraciones. En esta tabla se pueden observar individuos con varias reglas duplicadas.

8 Conclusiones y líneas futuras

8.1 Conclusiones

Se presenta en esta tesis fin de máster un nuevo método evolutivo inspirado en el proceso de la sedimentación para la obtención de bases de conocimiento con reglas difusas. Este método calcula, con el paso de las generaciones, un sustrato de reglas en la población que resultan ser muy beneficiosas para la solución final.

El método de la sedimentación emplea 3 nuevos operadores: cruce, adición y reducción. El operador de cruce mantiene las reglas comunes de los padres en los hijos, y reparte entre los descendientes las que son diferentes. Este operador elimina la posibilidad de que aparezcan bases de conocimiento con reglas duplicadas. También establece el sustrato de reglas óptimas que forma parte de la solución final del algoritmo evolutivo. El operador de adición añade una regla generada de forma aleatoria al individuo seleccionado. Este operador aumenta la diversidad genética del individuo y, por tanto, de la población. También permite explorar individuos con un mayor número de reglas. El operador de reducción elimina una regla al azar de la base de reglas del individuo elegido. Este operador permite explorar soluciones con un menor número de reglas. Con el paso de las generaciones, este operador sirve para eliminar reglas poco óptimas que estén muy extendidas en la población.

En este trabajo también se ha puesto de manifiesto el sesgo intrínseco que tiene el operador de cruce Whigham en la PGGG, y cómo afecta a los resultados obtenidos en función de la gramática empleada. Un diseño erróneo de la gramática puede producir resultados poco adecuados o, incluso, a la no convergencia del algoritmo. Se ha observado empíricamente que el método de la sedimentación obtiene soluciones con valores de adaptación similares al algoritmo PGGG con cruce Whigham, si se utiliza una gramática especialmente diseñada para evitar el sesgo observado. Si la gramática no se diseña con este propósito, los resultados son muy inferiores en comparación a los obtenidos por la sedimentación.

Los resultados muestran que el método de la sedimentación explora las posibles soluciones de manera más uniforme, empleando todas las variables de entrada al sistema. Como el método de la sedimentación evita el sesgo observado en el cruce Whigham y en la generación de individuos aleatorios, se favorece la búsqueda de un número más amplio y variado de soluciones sin disminuir su nivel de adaptación.

El nuevo método evolutivo se ha usado para obtener una base de conocimiento de reglas difusas que juegue a un videojuego de lucha 2D. El mayor número de antecedentes empleados en las reglas obtenidas mediante sedimentación hace que se

obtengan resultados que combinan múltiples entradas al sistema de forma óptima, lo que permite una mayor adaptabilidad en situaciones distintas a las del entrenamiento. Los algoritmos basados en PGGG se limitan a obtener reglas con un único antecedente, de forma que las reglas más complejas quedan sin explorar.

Los resultados muestran que el método de la sedimentación consigue buenas soluciones independientemente de la gramática empleada para describir las reglas de la base de conocimiento. En los problemas más complejos donde la gramática es muy extensa, el método de la sedimentación tiene la ventaja de que permite mayor flexibilidad a la hora de diseñarla sin perder calidad en las soluciones encontradas.

Los datos obtenidos muestran la capacidad de exploración del método de la sedimentación. En las soluciones basadas en la PGGG se ha observado su tendencia a emplear un reducido conjunto de las variables de entrada en las reglas obtenidas. El algoritmo evolutivo basado en la sedimentación obtiene individuos con una mayor variedad de variables de entrada. Esto permite obtener una mejor experiencia de juego y una mejor adaptabilidad ante diferentes situaciones.

8.2 Líneas futuras

Las líneas futuras de investigación que surgen como consecuencia del desarrollo del método de la sedimentación y observación del sesgo producido en la PGGG en el operador de cruce Whigham y la generación de individuos de forma aleatoria, se puede resumir en los siguientes.

El método de la sedimentación se ha usado en un problema concreto, generar bases de conocimiento de reglas difusas para juegos de lucha 2D. Sería interesante estudiar la posibilidad de que este método fuera de propósito general. El estudio debería abarcar un mayor número de problemas donde la solución esté basada en reglas. Adicionalmente, sería interesante comprobar la escalabilidad del método, aumentando el número de antecedentes y consecuentes posibles.

Se define la granularidad en un algoritmo evolutivo como el tamaño mínimo en el que se pueden dividir las partes que componen un individuo [XING09]. La PGGG tiene una granularidad fina debido a que trabaja a nivel de símbolo. Se considera que el método de la sedimentación tiene una granularidad gruesa al trabajar a nivel de regla. Pese a haber observado que no supone un gran problema a la hora de encontrar soluciones óptimas para el problema objetivo de este trabajo, una posible mejora

consistiría en intentar disminuir la granularidad a nivel de antecedentes, de consecuentes y de los valores que toman. De este modo se espera conseguir una convergencia más rápida del algoritmo puesto que elimina la aleatoriedad completa en la generación del contenido de las reglas.

9 Anexos

9.1 Anexo 1

Gramática no optimizada para evitar el sesgo por el cruce Whigham, también denominada gramática en cascada por la forma que adquieren sus árboles de derivación.

```
#A# S
#N# REGLAS REGLA CONDICIONES ACCION CONDICION CONDICION_DISTANCIA
VALORES_DISTANCIA USO_SKILL SKILL COMBO MOVIMIENTO
MOVIMIENTO_HORIZONTAL MOVIMIENTO_VERTICAL VALORES_HORIZONTAL
VALORES_VERTICAL CONDICION_ENERGIA_OWN CONDICION_ENERGIA_OPP
VALORES_ENERGIA CONDICION_ACCION_OPP VALORES_OPP_ACCION
CONDICION_POSITION_OWN CONDICION_POSITION_OPP VALORES_POSITION
CONDICION_PUNTUACION VALORES_PUNTUACION CONDICIONES_1
CONDICIONES_2 CONDICIONES_3 CONDICIONES_4 CONDICIONES_5
CONDICIONES_6
#T# ; if then and is DISTANCIA VERY_FAR FAR CLOSE VERY_CLOSE TRY
DO_IT HorizontalMove BACK STAND FORWARD VerticalMove CROUCH STAND
JUMP OWN_ENERGY OPP_ENERGY NONE LOW GOOD FULL NEUTRAL STAND
FORWARD_WALK DASH BACK_STEP CROUCH JUMP FOR_JUMP BACK_JUMP AIR
STAND_GUARD CROUCH_GUARD AIR_GUARD STAND_GUARD_RECOV
CROUCH_GUARD_RECOV AIR_GUARD_RECOV STAND_RECOV CROUCH_RECOV
AIR_RECOV CHANGE_DOWN DOWN RISE LANDING THROW_A THROW_B THROW_HIT
THROW_SUFFER STAND_A STAND_B CROUCH_A CROUCH_B AIR_A AIR_B AIR_DA
AIR_DB STAND_FA STAND_FB CROUCH_FA CROUCH_FB AIR_FA AIR_FB AIR_UA
AIR_UB STAND_D_DF_FA STAND_D_DF_FB STAND_F_D_DFA STAND_F_D_DFB
STAND_D_DB_BA STAND_D_DB_BB AIR_D_DF_FA AIR_D_DF_FB AIR_F_D_DFA
AIR_F_D_DFB AIR_D_DB_BA AIR_D_DB_BB STAND_D_DF_FC OPP_ACTION
OWN_POSITION OPP_POSITION CENTER SIDE SCORE LOSE DRAW WIN
S ::= REGLAS
REGLAS ::= REGLA | REGLAS ; REGLA
REGLA ::= if CONDICIONES then ACCION
CONDICIONES ::= CONDICION_PUNTUACION | CONDICION_PUNTUACION and
CONDICIONES_1 | CONDICIONES_1
CONDICIONES_1 ::= CONDICION_ENERGIA_OWN | CONDICION_ENERGIA_OWN
and CONDICIONES_2 | CONDICIONES_2
CONDICIONES_2 ::= CONDICION_ENERGIA_OPP | CONDICION_ENERGIA_OPP
and CONDICIONES_3 | CONDICIONES_3
CONDICIONES_3 ::= CONDICION_ACCION_OPP | CONDICION_ACCION_OPP and
CONDICIONES_4 | CONDICIONES_4
CONDICIONES_4 ::= CONDICION_POSITION_OWN | CONDICION_POSITION_OWN
and CONDICIONES_5 | CONDICIONES_5
CONDICIONES_5 ::= CONDICION_POSITION_OPP | CONDICION_POSITION_OPP
and CONDICIONES_6 | CONDICIONES_6
CONDICIONES_6 ::= CONDICION_DISTANCIA

CONDICION_DISTANCIA ::= DISTANCIA is VALORES_DISTANCIA
VALORES_DISTANCIA ::= VERY_FAR | FAR | CLOSE | VERY_CLOSE
CONDICION_ENERGIA_OWN ::= OWN_ENERGY is VALORES_ENERGIA
```

```
CONDICION_ENERGIA_OPP ::= OPP_ENERGY is VALORES_ENERGIA
VALORES_ENERGIA ::= NONE | LOW | GOOD | FULL
CONDICION_ACCION_OPP ::= OPP_ACTION is VALORES_OPP_ACCION
VALORES_OPP_ACCION ::= NEUTRAL | STAND | FORWARD_WALK | DASH |
BACK_STEP | CROUCH | JUMP | FOR_JUMP | BACK_JUMP | AIR |
STAND_GUARD | CROUCH_GUARD | AIR_GUARD | STAND_GUARD_RECOV |
CROUCH_GUARD_RECOV | AIR_GUARD_RECOV | STAND_RECOV | CROUCH_RECOV
| AIR_RECOV | CHANGE_DOWN | DOWN | RISE | LANDING | THROW_A |
THROW_B | THROW_HIT | THROW_SUFFER | STAND_A | STAND_B | CROUCH_A
| CROUCH_B | AIR_A | AIR_B | AIR_DA | AIR_DB | STAND_FA | STAND_FB
| CROUCH_FA | CROUCH_FB | AIR_FA | AIR_FB | AIR_UA | AIR_UB |
STAND_D_DF_FA | STAND_D_DF_FB | STAND_F_D_DFA | STAND_F_D_DFB |
STAND_D_DB_BA | STAND_D_DB_BB | AIR_D_DF_FA | AIR_D_DF_FB |
AIR_F_D_DFA | AIR_F_D_DFB | AIR_D_DB_BA | AIR_D_DB_BB |
STAND_D_DF_FC
CONDICION_POSITION_OWN ::= OWN_POSITION is VALORES_POSITION
CONDICION_POSITION_OPP ::= OPP_POSITION is VALORES_POSITION
VALORES_POSITION ::= CENTER | SIDE
CONDICION_PUNTUACION ::= SCORE is VALORES_PUNTUACION
VALORES_PUNTUACION ::= LOSE | DRAW | WIN
ACCION ::= COMBO | MOVIMIENTO
COMBO ::= SKILL is USO_SKILL
SKILL ::= STAND_GUARD | CROUCH_GUARD | AIR_GUARD | THROW_A |
THROW_B | STAND_A | STAND_B | CROUCH_A | CROUCH_B | AIR_A | AIR_B
| AIR_DA | AIR_DB | STAND_FA | STAND_FB | CROUCH_FA | CROUCH_FB |
AIR_FA | AIR_FB | AIR_UA | AIR_UB | STAND_D_DF_FA | STAND_D_DF_FB
| STAND_F_D_DFA | STAND_F_D_DFB | STAND_D_DB_BA | STAND_D_DB_BB |
AIR_D_DF_FA | AIR_D_DF_FB | AIR_F_D_DFA | AIR_F_D_DFB |
AIR_D_DB_BA | AIR_D_DB_BB | STAND_D_DF_FC
USO_SKILL ::= TRY | DO_IT
MOVIMIENTO ::= MOVIMIENTO_HORIZONTAL | MOVIMIENTO_VERTICAL
MOVIMIENTO_HORIZONTAL ::= HorizontalMove is VALORES_HORIZONTAL
VALORES_HORIZONTAL ::= BACK | STAND | FORWARD
MOVIMIENTO_VERTICAL ::= VerticalMove is VALORES_VERTICAL
VALORES_VERTICAL ::= CROUCH | STAND | JUMP
```

9.2 Anexo 2

Gramática optimizada para evitar el sesgo por cruce Whigham. La PGGG que usa esta gramática es la que consiguió mejores resultados de las diferentes gramáticas empleadas.

```
#A# S
#N# REGLAS REGLA CONDICIONES ACCION CONDICION CONDICION_DISTANCIA
VALORES_DISTANCIA USO_SKILL SKILL COMBO MOVIMIENTO
MOVIMIENTO_HORIZONTAL MOVIMIENTO_VERTICAL VALORES_HORIZONTAL
VALORES_VERTICAL CONDICION_ENERGIA VALORES_ENERGIA
```

```
CONDICION_ACCION_OPP VALORES_OPP_ACCION CONDICION_POSITION
VALORES_POSITION CONDICION_PUNTUACION VALORES_PUNTUACION
CONDICIONESIZ CONDICIONESDER CONDICIONESIZ_I
#T# ; if then and is DISTANCIA VERY_FAR FAR CLOSE VERY_CLOSE TRY
DO_IT HorizontalMove BACK STAND FORWARD VerticalMove CROUCH STAND
JUMP OWN_ENERGY OPP_ENERGY NONE LOW GOOD FULL NEUTRAL STAND
FORWARD_WALK DASH BACK_STEP CROUCH JUMP FOR_JUMP BACK_JUMP AIR
STAND_GUARD CROUCH_GUARD AIR_GUARD STAND_GUARD_RECOV
CROUCH_GUARD_RECOV AIR_GUARD_RECOV STAND_RECOV CROUCH_RECOV
AIR_RECOV CHANGE_DOWN DOWN RISE LANDING THROW_A THROW_B THROW_HIT
THROW_SUFFER STAND_A STAND_B CROUCH_A CROUCH_B AIR_A AIR_B AIR_DA
AIR_DB STAND_FA STAND_FB CROUCH_FA CROUCH_FB AIR_FA AIR_FB AIR_UA
AIR_UB STAND_D_DF_FA STAND_D_DF_FB STAND_F_D_DFA STAND_F_D_DFB
STAND_D_DB_BA STAND_D_DB_BB AIR_D_DF_FA AIR_D_DF_FB AIR_F_D_DFA
AIR_F_D_DFB AIR_D_DB_BA AIR_D_DB_BB STAND_D_DF_FC OPP_ACTION
OWN_POSITION OPP_POSITION CENTER SIDE SCORE LOSE DRAW WIN
S ::= REGLAS
REGLAS ::= REGLA | REGLAS ; REGLA
REGLA ::= if CONDICIONES then ACCION
CONDICIONES ::= CONDICIONESIZ | CONDICIONESIZ and CONDICIONESDER |
CONDICIONESDER
CONDICIONESIZ ::= CONDICIONESIZ_I | CONDICIONESIZ_I and
CONDICION_ACCION_OPP | CONDICION_ACCION_OPP
CONDICIONESIZ_I ::= CONDICION_DISTANCIA | CONDICION_DISTANCIA and
CONDICION_ENERGIA | CONDICION_ENERGIA
CONDICIONESDER ::= CONDICION_POSITION | CONDICION_POSITION and
CONDICION_PUNTUACION | CONDICION_PUNTUACION
CONDICION_DISTANCIA ::= DISTANCIA is VALORES_DISTANCIA
VALORES_DISTANCIA ::= VERY_FAR | FAR | CLOSE | VERY_CLOSE
CONDICION_ENERGIA ::= OWN_ENERGY is VALORES_ENERGIA | OPP_ENERGY
is VALORES_ENERGIA
VALORES_ENERGIA ::= NONE | LOW | GOOD | FULL
CONDICION_ACCION_OPP ::= OPP_ACTION is VALORES_OPP_ACCION
VALORES_OPP_ACCION ::= NEUTRAL | STAND | FORWARD_WALK | DASH |
BACK_STEP | CROUCH | JUMP | FOR_JUMP | BACK_JUMP | AIR |
STAND_GUARD | CROUCH_GUARD | AIR_GUARD | STAND_GUARD_RECOV |
CROUCH_GUARD_RECOV | AIR_GUARD_RECOV | STAND_RECOV | CROUCH_RECOV
| AIR_RECOV | CHANGE_DOWN | DOWN | RISE | LANDING | THROW_A |
THROW_B | THROW_HIT | THROW_SUFFER | STAND_A | STAND_B | CROUCH_A
| CROUCH_B | AIR_A | AIR_B | AIR_DA | AIR_DB | STAND_FA | STAND_FB
| CROUCH_FA | CROUCH_FB | AIR_FA | AIR_FB | AIR_UA | AIR_UB |
STAND_D_DF_FA | STAND_D_DF_FB | STAND_F_D_DFA | STAND_F_D_DFB |
STAND_D_DB_BA | STAND_D_DB_BB | AIR_D_DF_FA | AIR_D_DF_FB |
AIR_F_D_DFA | AIR_F_D_DFB | AIR_D_DB_BA | AIR_D_DB_BB |
STAND_D_DF_FC
CONDICION_POSITION ::= OWN_POSITION is VALORES_POSITION|
OPP_POSITION is VALORES_POSITION
VALORES_POSITION ::= CENTER | SIDE
CONDICION_PUNTUACION ::= SCORE is VALORES_PUNTUACION
VALORES_PUNTUACION ::= LOSE | DRAW | WIN
ACCION ::= COMBO | MOVIMIENTO
COMBO ::= SKILL is USO_SKILL
SKILL ::= STAND_GUARD | CROUCH_GUARD | AIR_GUARD | THROW_A |
THROW_B | STAND_A | STAND_B | CROUCH_A | CROUCH_B | AIR_A | AIR_B
| AIR_DA | AIR_DB | STAND_FA | STAND_FB | CROUCH_FA | CROUCH_FB |
AIR_FA | AIR_FB | AIR_UA | AIR_UB | STAND_D_DF_FA | STAND_D_DF_FB
```

```
| STAND_F_D_DFA | STAND_F_D_DFB | STAND_D_DB_BA | STAND_D_DB_BB |
AIR_D_DF_FA | AIR_D_DF_FB | AIR_F_D_DFA | AIR_F_D_DFB |
AIR_D_DB_BA | AIR_D_DB_BB | STAND_D_DF_FC
USO_SKILL ::= TRY | DO_IT
MOVIMIENTO ::= MOVIMIENTO_HORIZONTAL | MOVIMIENTO_VERTICAL
MOVIMIENTO_HORIZONTAL ::= HorizontalMove is VALORES_HORIZONTAL
VALORES_HORIZONTAL ::= BACK | STAND | FORWARD
MOVIMIENTO_VERTICAL ::= VerticalMove is VALORES_VERTICAL
VALORES_VERTICAL ::= CROUCH | STAND | JUMP
```

9.3 Anexo 3

Bases de conocimiento obtenidas mediante los tres métodos utilizados. En la sedimentación se observan la variedad de reglas y de antecedentes empleados. En la PGGG con la gramática optimizada se observan las bases de reglas más reducidas donde la posición del oponente y la distancia prevalecen sobre el resto de antecedentes. En la PGGG no optimizada se pueden observar bases de conocimiento con varias reglas duplicadas.

Individuo	Sedimentación	PGGG optimizada	PGGG no optimizada
1	if DISTANCIA is CLOSE then AIR_DB is DO_IT	if DISTANCIA is CLOSE then STAND_F_D_DFB is TRY	if OPP_ENERGY is GOOD then CROUCH_FB is DO_IT if OPP_ENERGY is NONE then HorizontalMove is BACK if OPP_ENERGY is NONE then CROUCH_FB is DO_IT
2	if OPP_POSITION is CENTER then STAND_F_D_DFB is DO_IT if OPP_POSITION is SIDE then CROUCH_B is DO_IT	if OPP_POSITION is CENTER then AIR_DB is TRY if SCORE is LOSE then AIR_DB is TRY if OPP_POSITION is SIDE then AIR_DB is TRY if OPP_POSITION is CENTER then AIR_DB is TRY	if OPP_ENERGY is FULL then AIR_DB is DO_IT if OPP_ENERGY is NONE then AIR_DB is DO_IT if OWN_ENERGY is LOW and OPP_ACTION is STAND_GUARD_RECOV and OWN_POSITION is CENTER then AIR_DB is DO_IT if OPP_ENERGY is GOOD then AIR_DB is DO_IT
3	if DISTANCIA is CLOSE then STAND_F_D_DFB is TRY if DISTANCIA is VERY_CLOSE then AIR_DB is TRY if DISTANCIA is VERY_CLOSE and OPP_ACTION is AIR_RECOV and OPP_ENERGY is NONE and OPP_POSITION is CENTER and OWN_ENERGY is NONE and	if DISTANCIA is CLOSE then STAND_F_D_DFB is TRY if OPP_POSITION is CENTER then STAND_F_D_DFB is TRY	if OPP_ENERGY is NONE then STAND_F_D_DFB is TRY if OPP_ENERGY is FULL then AIR_DB is TRY if OPP_ENERGY is NONE then AIR_DB is TRY if OPP_ENERGY is LOW then STAND_F_D_DFB is TRY if OPP_ENERGY is GOOD then AIR_DB is TRY if OPP_ENERGY is LOW then STAND_F_D_DFB is TRY if OPP_ENERGY is NONE then STAND_F_D_DFB is TRY

	OWN_POSITION is SIDE and SCORE is WIN then VerticalMove is STAND if DISTANCIA is VERY_FAR and OPP_ACTION is AIR_D_DB_BA and OPP_ENERGY is GOOD and OPP_POSITION is SIDE and OWN_ENERGY is GOOD and OWN_POSITION is CENTER then VerticalMove is STAND		if SCORE is DRAW then AIR_DB is TRY

10 Bibliografía

[ACTI09] http://investor.activision.com/releasedetail.cfm?ReleaseID=426784 (2009). Último acceso: 06/2015

[ANIB04] Valenzuela, L. A., Bentley, J. M., & Lorenz, R. D. (2004). Expert system for integrated control and supervision of dry-end sections of paper machines. Industry Applications, IEEE Transactions on, 40(2), 680-691.

[AVER11] Avery, P., Togelius, J., Alistar, E., & Van Leeuwen, R. P. (2011, June). Computational intelligence and tower defence games. In *Evolutionary Computation (CEC), 2011 IEEE Congress on* (pp. 1084-1091). IEEE.

[BARR91] Barrios, D. (1991). "Operador de Cruce Generalizado en Algoritmos Genéticos". Tesis Doctoral, Facultad de Informática, Universidad Politécnica de Madrid.

[BOUR04] Bourg, D. M., & Seemann, G. (2004). AI for game developers. " O'Reilly Media, Inc.".

[BROW08] Browne, C. (2008). *Automatic generation and evaluation of recombination games* (Doctoral dissertation, Queensland University of Technology).

[CHOM57] Chomsky, N. (1957). *Syntactic structures*. Walter de Gruyter.

[CORD01] Cordón, O., Herrera, F., Gomide, F., Hoffmann, F., & Magdalena, L. (2001, July). Ten years of genetic fuzzy systems: current framework and new trends. In *IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th* (Vol. 3, pp. 1241-1246). IEEE.

[CHAR29] Charles, D. (1929). *The origin of species*. London: John Murry.

[DORA00] Dorado, J. (2000). *Modelo de un sistema para la selección automática en dominios complejos, con una estrategia cooperativa, de conjuntos de entrenamiento y arquitecturas ideales de redes de neuronas artificiales utilizando algoritmos genéticos* (Doctoral dissertation, Universidade da Coruña).

[DORI10] Dorigo, M., & Birattari, M. (2010). Ant colony optimization. In *Encyclopedia of Machine Learning* (pp. 36-39). Springer US.

[EBNE09] Ebner, M., & Tiede, T. (2009, September). Evolving driving controllers using genetic programming. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on* (pp. 279-286). IEEE.

[ENGE07] Engelbrecht, A. P. (2007). Computational intelligence: an introduction. John Wiley & Sons.

[FOGE66] Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). Artificial intelligence through simulated evolution.

[FONT09] Font, J. M., Manrique, D., & Ríos, J. (2009). Redes de Neuronas Artificiales y Computación Evolutiva. Fundación General de la Univ. Politécnica de Madrid.

[FONT12] Font, J. M. (2012). Evolving third-person shooter enemies to optimize player satisfaction in real-time. In *Applications of Evolutionary Computation* (pp. 204-213). Springer Berlin Heidelberg.

[FUZZ15] http://www.fuzzylite.com/ Versión 4.0 (2015). Último acceso: 06/2015

[GOET95] Goethe, J. W., & Bronzino, J. D. (1995). An expert system for monitoring psychiatric treatment. Engineering in Medicine and Biology Magazine, IEEE, 14(6), 776-780.

[HAIG09] Haigh-Hutchinson, M. (2009). Real Time Cameras: A Guide for Game Designers and Developers. Morgan Kaufmann Publishers Inc..

[HERN88] Hernández, J. L. M., & Sierra, J. P. (1988). Ingeniería del conocimiento: Diseño y construcción de sistemas expertos.

[HOLL69] Holland, J. H. (1969). *ADAPTIVE PLANS OPTIMAL FOR PAYOFF-ONLY ENVIRONMENTS* (No. 08226-10-T). MICHIGAN UNIV ANN ARBOR LOGIC OF COMPUTERS GROUP.

[HOLL92] John Henry Holland. (1992). Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press.

[IANM06] Ian, M. (2006). Artificial Intelligence for Games.

[ICEC15]http://www.ice.ci.ritsumei.ac.jp/~ftgaic/Downloadfiles/Brief%20table%20of%20ZEN's%20skills.pdf (2015)

[ISAS97] Isasi, P., Martínez P., Borrajo, D. (1997) *Lenguajes, Gramáticas y Autómatas. Un enfoque práctico*. Addison-Wesley.

[JANG12] S.S. Jang, K.T. Oishi, R.G. Egbert, and E. Klavins, (2012) *Specification and simulation of multicelled behaviors*, ACS Synthetic Biology,.

[JENN10] Jennings-Teats, M., Smith, G., & Wardrip-Fruin, N. (2010, October). Polymorph: A model for dynamic level generation. In *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[JERR77] Jerri, A. J. (1977). The Shannon sampling theorem—Its various extensions and applications: A tutorial review. Proceedings of the IEEE, 65(11), 1565-1596.

[KARI08] Kari, L., & Rozenberg, G. (2008). The many facets of natural computing. Communications of the ACM, 51(10), 72-83.

[KARP87] Karp, R. M., & Rabin, M. O. (1987). Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development, 31(2), 249-260.

[KENN10] Kennedy, J. (2010). Particle swarm optimization. In *Encyclopedia of Machine Learning* (pp. 760-766). Springer US.

[KOST13] Koster, R. (2013). *Theory of fun for game design*. " O'Reilly Media, Inc.".

[KOZA92] Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection* (Vol. 1). MIT press.

[LIAP13] Liapis, A., Yannakakis, G. N., & Togelius, J. (2013, November). Towards a Generic Method of Evaluating Game Levels. In *AIIDE*.

[MAHA03] Mahaman, B. D., Passam, H. C., Sideridis, A. B., & Yialouris, C. P. (2003). DIARES-IPM: a diagnostic advisory rule-based expert system for integrated pest management in Solanaceous crop systems. Agricultural Systems, 76(3), 1119-1135.

[MAMD75] Mamdani, E. H., & Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. International journal of man-machine studies, 7(1), 1-13.

[MANR01] Manrique, D. (2001). Diseño de Redes de Neuronas y Nuevas Técnicas de Optimización mediante Algoritmos Genéticos (Doctoral dissertation, Tesis Doctoral). Universidad Politécnica de Madrid.

[MARC92]Marcos, A. (1992). Neodarwinismo, teoría de la información y termodinámica: estado de la cuestión. Estudios Filosóficos, 41, 215-252.

[MARI95] Marín, F. J., & Sandoval, F. (1995). Diseño de redes neuronales artificiales mediante algoritmos genéticos.

[MEIR91] Myerson, Roger B. (1991). Game Theory: Analysis of Conflict, Harvard University Press,

[MORA08] Morales, R. L. M., & Méndez, J. T. P. (2008). Inteligencia artificial: técnicas, métodos y aplicaciones.

[OBRI96] O'Brien, L. (1996). Fuzzy logic in games. Game Developer Magazine, 3(2), 52-55.

[PAJA05] Pajares, G., Santos, M. (2005). Inteligencia Artificial e Ingeniería del Conocimiento. Ra-Ma.

[PERE09] Perez, D., Recio, G., Saez, Y., & Isasi, P. (2009, September). Evolving a fuzzy controller for a car racing competition. In Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on (pp. 263-270). IEEE.

[PIRO12] Pirovano, M. (2012). The use of Fuzzy Logic for Artificial Intelligence in Games. University of Milano, Milano.

[POWE02] Power, D. J., Sharda, R., & Burstein, F. (2002). Decision support systems. John Wiley & Sons, Ltd.

[RECH65] Rechenberg, I. (1965). Cybernetic solution path of an experimental Problem",(Royal aircraft establishment translation no. 1122, BF toms, trans.). *Farnsborough Hants: Ministery of Aviation, Royal Aircraft Establishment*, *1122*.

[RUSS95] Russel, S., Norvig, P., (1995). Artificial Intelligence: A Modern Approach, Prentice-Hall, Inc.

[SCHM06] Schmidhuber, J. (2006). Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, *18*(2), 173-187.

[SCHW09] Schwab, B. (2009). AI game engine programming. Cengage Learning.

[SORE10] Sorenson, N., & Pasquier, P. (2010). Towards a generic framework for automated video game level creation. In *Applications of Evolutionary Computation* (pp. 131-140). Springer Berlin Heidelberg.

[SORE11] Sorenson, N., Pasquier, P., & DiPaola, S. (2011). A generic approach to challenge modeling for the procedural creation of video game levels. *Computational Intelligence and AI in Games, IEEE Transactions on*, *3*(3), 229-244.

[TANA92] Tanaka, K., & Sugeno, M. (1992). Stability analysis and design of fuzzy control systems. Fuzzy sets and systems, 45(2), 135-156.

[TOGE10] Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelback, J., & Yannakakis, G. N. (2010, August). Multiobjective exploration of the starcraft map

space. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on* (pp. 265-272). IEEE.

[TOGE11] Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, *3*(3), 172-186.

[TRIL92] Trillas, E., & Ríos, J. G. (1992). *Aplicaciones de la lógica borrosa* (Vol. 20). Editorial CSIC-CSIC Press.

[TVTR09] http://tvtropes.org/pmwiki/pmwiki.php/Main/PerfectPlayAI (2009). Último acceso: 06/2015

[USGM13] http://www.usgamer.net/articles/top-10-biggest-grossing-arcade-games-of-all-time (2013). Último acceso: 06/2015

[WHI95] Whigham, P. A. (1995). Inductive bias and genetic programming.

[XING09] Xing, H., Liu, X., Jin, X., Bai, L., & Ji, Y. (2009). A multi-granularity evolution based Quantum Genetic Algorithm for QoS multicast routing problem in WDM networks. Computer Communications, 32(2), 386-393.

[ZADE65] Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, *8*(3), 338-353.